2-2010

Basis Construction and Utilization for Markov Decision Processes Using Graphs

Jeffrey Thomas Johns
University of Massachusetts Amherst, jefftjohns@yahoo.com

BASIS CONSTRUCTION AND UTILIZATION FOR MARKOV DECISION PROCESSES USING GRAPHS

A Dissertation Presented

by

JEFFREY T. JOHNS

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

February 2010

Department of Computer Science

BASIS CONSTRUCTION AND UTILIZATION FOR MARKOV DECISION PROCESSES USING GRAPHS

A Dissertation Presented

by

JEFFREY T. JOHNS

Approved as to style and content by:

Sridhar Mahadevan, Chair

Andrew G. Barto, Member

Shlomo Zilberstein, Member

John Staudenmayer, Member

Andrew G. Barto, Department Chair
Department of Computer Science

To Emerson and Marlene Johns for caring as much as they do.

ACKNOWLEDGMENTS

I would like to thank my thesis advisor, Sridhar Mahadevan. Sridhar has cultivated a working environment in which I have felt free to explore many different ideas and research topics. I enjoyed this sense of academic freedom and am grateful for Sridhar's experience in knowing when to focus my efforts in a particular area. I am also very appreciative of the feedback and support offered by my thesis committee, Andrew Barto, Shlomo Zilberstein, and John Staudenmayer.

I am fortunate to have worked on an education project with Beverly Woolf and Ivon Arroyo. It was a pleasure (and a challenge) thinking of ways to help high school students improve their math skills.

I am grateful for the wonderful people in the Autonomous Learning Laboratory. I especially thank Sarah Osentoski. Our camaraderie, both personally and professionally, has helped make the journey through graduate school fun. I have enjoyed and benefited from many interesting research discussions with George Konidaris and Marek Petrik. George's enthusiasm and intellectual curiosity are infectious. I thank Anders Jonsson for writing a letter of recommendation that helped me, without a computer science degree, get into graduate school. I would also like to recognize Ashvin Shah, Andrew Stout, Güray Alsaç, Chang Wang, Chris Vigorito, Colin Barringer, Kimberly Ferguson, and Pippin Wolfe, who have all helped influence my research perspective.

My parents, Emerson and Marlene Johns, have been my biggest supporters. They have encouraged my pursuits and instilled in me the confidence to accomplish them. I am truly blessed and thankful to have them in my corner. I am also grateful for family and friends who have kept me grounded while immersed in my studies.

I have been fortunate during my time at the University of Massachusetts to have been supported by the National Science Foundation under grants IIS-0534999 and IIS-0803288. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

ABSTRACT

BASIS CONSTRUCTION AND UTILIZATION FOR MARKOV DECISION PROCESSES USING GRAPHS

FEBRUARY 2010

JEFFREY T. JOHNS

B.Sc., UNIVERSITY OF VIRGINIA

M.Sc., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Sridhar Mahadevan

The ease or difficulty in solving a problem strongly depends on the way it is represented. For example, consider the task of multiplying the numbers 12 and 24. Now imagine multiplying XII and XXIV. Both tasks can be solved, but it is clearly more difficult to use the Roman numeral representations of twelve and twenty-four. Humans excel at finding appropriate representations for solving complex problems. This is not true for artificial systems, which have largely relied on humans to provide appropriate representations. The ability to *autonomously* construct useful representations and to efficiently exploit them is an important challenge for artificial intelligence.

This dissertation builds on a recently introduced graph-based approach to learning representations for sequential decision-making problems modeled as Markov decision processes (MDPs). Representations, or basis functions, for MDPs are abstractions of the problem's state space and are used to approximate value functions, which quantify the expected

long-term utility obtained by following a policy. The graph-based approach generates basis functions capturing the structure of the environment. Handling large environments requires efficiently *constructing* and *utilizing* these functions. We address two issues with this approach: (1) scaling basis construction and value function approximation to large graphs/data sets, and (2) tailoring the approximation to a specific policy's value function.

We introduce two algorithms for computing basis functions from large graphs. Both algorithms work by decomposing the basis construction problem into smaller, more manageable subproblems. One method determines the subproblems by enforcing block structure, or groupings of states. The other method uses recursion to solve subproblems which are then used for approximating the original problem. Both algorithms result in a set of basis functions from which we employ basis selection algorithms. The selection algorithms represent the value function with as few basis functions as possible, thereby reducing the computational complexity of value function approximation and preventing overfitting.

The use of basis selection algorithms not only addresses the scaling problem but also allows for tailoring the approximation to a specific policy. This results in a more accurate representation than obtained when using the same subset of basis functions irrespective of the policy being evaluated. To make effective use of the data, we develop a hybrid least-squares algorithm for setting basis function coefficients. This algorithm is a parametric combination of two common least-squares methods used for MDPs. We provide a geometric and analytical interpretation of these methods and demonstrate the hybrid algorithm's ability to discover improved policies. We also show how the algorithm can include graph-based regularization to help with sparse samples from stochastic environments.

This work investigates all aspects of linear value function approximation: constructing a dictionary of basis functions, selecting a subset of basis functions from the dictionary, and setting the coefficients on the selected basis functions. We empirically evaluate each of these contributions in isolation and in one combined architecture.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Motivation

Herbert Simon [90] wrote, "solving a problem simply means representing it so as to make the solution transparent." This statement underscores the importance of *representations* and, in particular, how the right representation can make problems simple. This is particularly true for artificial intelligence agents that must effectively operate in complex, stochastic environments. Although some knowledge can be handcrafted by those designing the agents, truly autonomous agents must be able to construct their own representations to deal with unique and unforeseen problems they invariably will encounter. But how can an agent construct its own representations?

We consider the problem of representation discovery in the context of sequential decision making. Stochastic sequential decision making problems are most often studied using the mathematical formalism of Markov decision processes (MDPs) [85]. In this framework, an agent chooses which action to perform thereby causing the agent to transition (stochastically) from one state of the environment to another state. The agent receives a numerical reward for each transition. The agent's goal is to learn which action to perform in each state (i.e. a policy) so as to maximize expected *cumulative* reward. To help make these decisions, an agent can estimate the value of states in the environment. The agent's *representation* of the value of a state strongly influences its behavior.

To ground this problem in a concrete example, consider the well-known game of Tetris. A standard Tetris board consists of 200 cells (20 columns, 10 rows) that are either occupied or vacant; therefore, there are $O(2^{200})$ possible states of the game. It is clearly infeasible

trying to learn a value for each unique state. Instead, by trial-and-error, humans have devised features that encode concepts like "column height" and "number of holes" [12]. The value of a Tetris state is then a weighted sum of the features (i.e., $w_1 \times (\text{column height}) + w_2 \times (\text{number of holes}) + \dots$, where $w_i \in \mathbb{R}, i = 1, 2, \dots$) where the agent is free to set the weights appropriately. This is known as linear value function approximation and the features are referred to as basis functions. Effective Tetris policies can be learned with these features [100], but the difficult task of abstracting the raw Tetris state space has been solved by humans rather than the agent. A flow diagram depicting this typical approach to solving complex MDPs is shown in Figure 1.1. It would be beneficial to remove the designer from this loop entirely and leave the agent with the challenging task of constructing useful features.

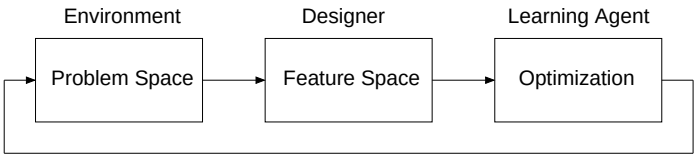
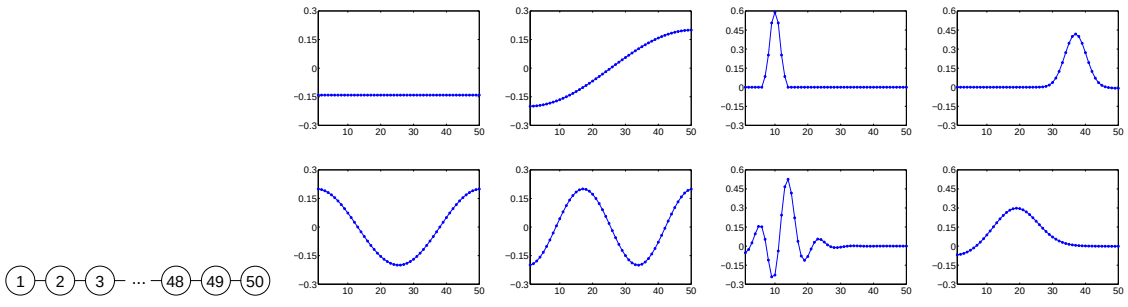


Figure 1.1. The typical framework for solving complex sequential decision making problems.

The importance of forming abstractions and generalizing experience in one setting to a different, but similar, setting is well known in the field of artificial intelligence (AI). Indeed, one of the earliest AI demonstrations dating back to 1959 was Arthur Samuel's famous checkers player [87]. Samuel used a polynomial function approximator to represent the value of a checkers board. Other common function approximators include radial basis functions, tile codings, and neural networks [98]. These are generic basis functions that do not (automatically) account for regularities that may exist in the agent's environment.

1.2 Overview

In this dissertation, we build on a recently introduced graph-based approach to automatically constructing representations for MDPs [67, 63]. Vertices in the graph correspond to states of the environment and edges connect similar states. Mahadevan and Maggioni proposed taking samples from a MDP, forming a graph from the samples, and computing either graph Laplacian eigenvectors [67] or diffusion wavelets [63] which are then used as basis functions. Laplacian eigenvectors, which have global support (i.e. every vertex in the graph has a value), are the *smoothest* eigenvectors of the Laplace operator (e.g. those with the smallest corresponding eigenvalue). Diffusion wavelets are localized functions at multiple scales. These basis functions capture the structure of the environment as modeled by the graph. Similar states in the graph have similar basis function values. Two strengths of the graph-based approach to basis construction are that it (1) creates basis functions that can be used for representing many functions over the state space, and (2) allows for modeling the *intrinsic* dimensionality of the environment. While an environment may nominally be very high dimensional, often times lower dimensional structure exists in the problem's dynamics. The graph-based approach leverages such structure and can thus circumvent the "curse of dimensionality." Figure 1.2 shows a chain graph as well as a few graph Laplacian eigenvectors and diffusion wavelets.



(a) Chain graph (b) Laplacian eigenvectors (c) Diffusion wavelet scaling functions

Figure 1.2. (a) 50 vertex chain graph, (b) first four graph Laplacian eigenvectors, and (c) four diffusion wavelet scaling functions at different scales.

This dissertation focuses on both basis construction and value function approximation. Our goal is to efficiently use a set of MDP samples to generate graph-based features and utilize those features to learn good policies. This requires (1) scaling basis construction and value function approximation to large graphs/data sets, and (2) tailoring the representation to fit specific value functions.

For the graph-based approach to basis construction to scale to large problems, we must address the computational efficiency of the algorithms. In the worst case, the graph-based methods have cubic complexity in the number of vertices in the graph. This calls into question the scalability of the methods to large graphs (and thus large state spaces). We propose two algorithms for generating basis functions that approximate those computed using exact methods defined in previous work [67, 63]. Both algorithms work by decomposing the basis construction problem into smaller, more manageable subproblems. The first algorithm we present uses the Kronecker product to automatically form basis construction subproblems. The Kronecker product is useful because it preserves certain properties associated with the original basis construction problem. We demonstrate how this method can significantly improve computational efficiency and memory usage. The second algorithm we present uses a multiscale recursive approach to basis construction. The state space graph is automatically decomposed into smaller subgraphs. This decomposition occurs recursively until the subgraphs are of a manageable size. Basis construction occurs on the subgraphs. The recursion then unwinds and the basis functions from the subgraphs are used to approximate the basis construction problem on the larger graphs. We compare these two algorithms empirically.

Given a set of MDP samples and the graph-based basis functions, we use least-squares algorithms for learning the basis function weights. Least-squares techniques are used to fit data to a model by minimizing an objective function. These techniques are important in the context of MDPs due to their sample efficiency. There are two common least-squares methods used for MDPs which behave very differently in practice. We develop a hybrid

least-squares algorithm that is a parametric combination of the two common methods' objective functions. A geometric interpretation of the hybrid least-squares algorithm along with empirical evidence suggests the hybrid algorithm can ultimately lead to the agent discovering better policies. We also extend the hybrid least-squares algorithm to include graph-based regularization. This form of regularization, which is used to prevent overfitting, helps to ensure the approximate value functions vary smoothly according to the graph structure.

We propose algorithms that tailor the graph-based basis functions to fit specific value functions. Prior work with graph-based features used a simple heuristic to determine which features to use when approximating a value function [67, 63, 68, 79, 44, 95]. The heuristic is to always use the smoothest basis functions according to the graph's structure. This mechanism is independent of the particular value function being estimated, meaning all value functions are estimated with the same set of features. This is a computationally simple technique and is robust to overfitting (although too much regularization is problematic), but it does not exploit the full power of the graph-based features. We treat the features as a dictionary from which a subset can be used to approximate any particular value function. This is important for two reasons. First, it allows for a more accurate representation of the value of a state. Second, by selecting a subset of features, we can use as few as possible which improves the computational efficiency of the hybrid least-squares algorithm. We evaluate four different basis selection algorithms. Comparing the performance of the different selection algorithms allows for an understanding of the challenges involved in combining value function approximation and basis selection.

To summarize, we make the following three contributions in this dissertation:

1. We derive a regularized hybrid least-squares algorithm for approximate policy evaluation. This is a sample efficient algorithm that combines two common least-squares methods used for MDPs. Experimental results demonstrate the hybrid algorithm's ability to discover improved policies.

2. We propose and evaluate two algorithms for scaling up graph-based basis construction to large problems. Both algorithms decompose the basis construction problem into smaller, more manageable basis construction subproblems. This improves the computational efficiency over previous methods.

3. Using the graph-based basis functions as a dictionary, we employ basis selection algorithms that tailor the representation to the specific policy being evaluated. We demonstrate this can lead to a more accurate representation of the value function provided the policy evaluation procedure remains stable.

When combined, these three contributions form a policy iteration framework that takes as input a set of samples from a MDP and ultimately outputs an approximate value function used to determine an agent's policy.

1.3 Outline

We provide the necessary background on Markov decision processes and reinforcement learning in Chapter 2. Value function approximation and different types of basis functions are also presented. Chapter 2 contains much of the notation used throughout this dissertation.

In Chapter 3, we describe how samples from a MDP can be used to form a state space graph. A detailed description of Laplacian eigenvectors and diffusion wavelets, two types of graph-based basis functions, is provided with illustrative examples.

We introduce hybrid least-squares algorithms for approximate policy evaluation in Chapter 4. Given a set of basis functions and samples from a MDP, these algorithms produce an approximate value function. Policy iteration experiments demonstrate the efficacy of hybrid methods. We also describe how the least-squares algorithm can employ graph-based regularization.

Chapter 5 proposes two techniques for scaling basis construction to large graphs. The utility of basis functions generated using these two techniques is determined empirically. In Chapter 6, we use the graph-based basis functions as a dictionary and evaluate several basis selection algorithms. Basis selection algorithms, which choose a subset of basis functions from the dictionary, tailor the representation to a particular value function. Chapter 7 summarizes the work presented in this dissertation. Conclusions and ideas for future work are discussed.

CHAPTER 2

BACKGROUND

This chapter introduces Markov decision processes and reinforcement learning, which provide the mathematical structure for studying sequential decision making. We also cover function approximation and least-squares methods as they apply to reinforcement learning.

2.1 Markov Decision Processes

A Markov decision process (MDP) is a mathematical model of sequential decision making under uncertainty [85]. A finite MDP is defined by a four-tuple $M = (S, A, P, R)$ where S is a set of states, A is a set of actions, P is a probability function with $P^a_{ss'}$ being the probability of transitioning from state s to state s' upon executing action a, and R is the reward function with $R^a_{ss'}$ being the expected immediate reward obtained as a result of taking action a in state s and transitioning to state s'. We use the notation A_s to refer to the set of admissible actions in state s. The Markov property dictates that future states of the MDP are conditionally independent of past states given the current state.

A policy π is a mapping from states to a probability distribution over actions. The value function V^π associated with policy π defines the expected long-term discounted sum of rewards. This is defined mathematically as

$$\begin{aligned} V^\pi(s) &= E_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\} \\ &= E_\pi\{r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s\} \\ &= \sum_{a \in A_s} \pi(s,a) \sum_{s' \in S} P^a_{ss'}(R^a_{ss'} + \gamma V^\pi(s')), \end{aligned}$$

where $E_\pi\{\cdot\}$ denotes the expected value by following policy π, r_t is the immediate reward received at time t, and $\gamma \in [0, 1)$ is a discount factor controlling the trade-off between immediate and long-term rewards. An optimal policy π^*, associated with the optimal value function V^*, has the property that $V^*(s) \geq V^{\pi'}(s)$ for every state $s \in S$ and any other policy π'. The optimal value function V^* is defined as follows:

$$\begin{aligned} V^*(s) &= \max_\pi V^\pi(s) \\ &= \max_{a \in A_s} \sum_{s' \in S} P^a_{ss'}(R^a_{ss'} + \gamma V^*(s')). \end{aligned}$$

A policy can be determined from a value function by performing one step of lookahead using the transition and reward models P and R and selecting the action with the highest expected value. Alternatively, a policy can be determined without relying on lookahead by defining an action-value function Q^π explicitly over state-action pairs. The action-value function $Q^\pi(s, a)$ is defined as the expected return when starting in state s, taking action a, and following policy π thereafter:

$$\begin{aligned} Q^\pi(s, a) &= E_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\} \\ &= E_\pi\{r_{t+1} + \gamma Q^\pi(s_{t+1}, \pi(s_{t+1})) | s_t = s, a_t = a\} \\ &= \sum_{s' \in S} P^a_{ss'}(R^a_{ss'} + \gamma \sum_{a' \in A_{s'}} \pi(s', a') Q^\pi(s', a')). \end{aligned}$$

Similarly, the optimal action-value function Q^* is defined as

$$\begin{aligned} Q^*(s, a) &= \max_\pi Q^\pi(s, a) \\ &= \sum_{s' \in S} P^a_{ss'}(R^a_{ss'} + \gamma \max_{a' \in A_{s'}} Q^*(s', a')). \end{aligned}$$

A policy can be determined easily from an action-value function without use of a model simply by selecting an action that has the largest Q value.

Broadly speaking, there are two approaches to learning a policy. The first approach estimates the value function (or the action-value function) and then derives a policy, whereas the second approach searches directly in the space of policies. There are many learning algorithms within each approach. In this dissertation, we focus on the value-based technique.

It is convenient to abbreviate the equations in this section using matrix notation. The value function can be represented as a vector $V^\pi \in \mathbb{R}^{|S|}$ where we assume an ordering of states such that:

$$V^\pi = \left[V^\pi(s_1), \ldots, V^\pi(s_{|S|})\right]^T.$$

The value function solves the Bellman equation:

$$V^\pi = R^\pi + \gamma P^\pi V^\pi := T^\pi(V^\pi),$$

where $T^\pi : \mathbb{R}^{|S|} \rightarrow \mathbb{R}^{|S|}$ is the Bellman operator. In this format, P^π is a matrix of size $|S| \times |S|$ with elements $P^\pi_{ij} = \sum_{a \in A_{s_i}} \pi(s_i, a) P^a_{s_i s_j}$ and R^π is a vector of dimensionality $|S|$ with elements $R^\pi_i = \sum_{a \in A_{s_i}} \pi(s_i, a) \sum_{s' \in S} P^a_{s_i s'} R^a_{s_i s'}$. All vectors are assumed to be column vectors. The action-value function can be expressed in a similar way with $Q^\pi \in \mathbb{R}^{|S||A|}$ where the actions are ordered such that:

$$Q^\pi = \left[Q^\pi(s_1, a_1), \ldots, Q^\pi(s_{|S|}, a_1), Q^\pi(s_1, a_2), \ldots, Q^\pi(s_{|S|}, a_{|A|})\right]^T.$$

We reuse notation and write the Bellman equation as:

$$Q^\pi = R^\pi + \gamma P^\pi Q^\pi := T^\pi(Q^\pi).$$

Note that in this case, the vector R^π and matrix P^π have dimension $|S||A|$, but are defined analogously as above. The dimensionality of R^π and P^π will be obvious from the context depending on whether value functions or action-value functions are being used.

2.2 Reinforcement Learning

The reinforcement learning (RL) [98] framework involves an agent interacting in an environment attempting to maximize a reward signal. The generality of this framework allows its application to a wide array of problems. For Markovian environments, RL problems can be modeled as MDPs. In the RL framework, the agent usually does not have access to the MDP model, but it receives samples from the model by interacting with the environment.

Many of the algorithms for solving RL problems are instances of temporal difference (TD) learning [96]. TD learning combines dynamic programming and Monte Carlo methods. The TD(0) algorithm estimates V^π using the following update upon transitioning from state s to s' given action $\pi(s)$ with reward r:

$$V(s) \leftarrow V(s) + \alpha \left[r + \gamma V(s') - V(s) \right]$$

where $\alpha \in (0, 1]$ is a step-size parameter. The estimated value $V(s)$ is updated based on the estimated value $V(s')$, i.e., TD(0) bootstraps. Under appropriate conditions, the TD(0) algorithm converges to the true value function V^π.

RL algorithms can also compute the optimal value function. The Q-learning algorithm [106] estimates the action-value function using the following update rule:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a' \in A_{s'}} Q(s', a') - Q(s, a) \right]$$

upon a taking action a, transitioning from state s to s', and receiving reward r. The Q-learning algorithm converges to Q^* under appropriate conditions.

2.3 Linear Value Function Approximation

An exact representation of the value function (action-value function) stores one value for every state (state-action pair). This representation is impractical for problems with

large, discrete state spaces or continuous state spaces. An intelligent agent must be able to use its experience to generalize to states it has never seen. Generalization can be accomplished using function approximation. Function approximation has been used extensively in reinforcement learning dating back to Arthur Samuel's famous checkers player [87] which used a polynomial approximator to represent the value of a checkers position.

A simple yet very useful architecture for approximating functions is to use a *linear* combination of basis functions.[1] For example, in the context of value function approximation, a linear approximation has the form

$$\hat{V}(s; w) = \sum_{j=1}^{K} \phi_j(s) w_j = \phi(s)^T w$$

where $\phi(s)$ is a length K state feature vector and w is a parameter vector of length K. The features of a state allow for generalization. For example, it is common for states that are *similar* to have similar feature vectors. Usually the number of features $K \ll |S|$ to ensure the representation is compact. The parameters w are adjusted during the course of learning to fit the value function using some error metric, such as mean-squared error (MSE). In matrix notation, the approximate value function is written $\hat{V} = [\phi_1, \phi_2, \ldots, \phi_k]\, w = \Phi w$ where $\Phi \in \mathbb{R}^{|S| \times K}$ is a basis function matrix.

There are other techniques for approximating functions using nonlinear architectures. One example is a neural network which maps inputs to outputs using nonlinear functions, such as the sigmoid or hyperbolic tangent functions. The advantages of the linear architecture are simplicity in terms of updating the weights w (which also facilitates theoretical analysis) and, when using MSE, there is only one setting of w that achieves the global minimum MSE (barring degenerate cases). This dissertation focuses exclusively on the linear function approximation architecture.

---

[1] Throughout this dissertation, we use the words *basis function*, *feature*, and *representation* interchangeably.

2.4 Basis Functions

The linear function approximation architecture maps states from the MDP to feature vectors $\phi(s) \in \mathbb{R}^K$. Each of the K components represents a feature or basis function. Given the basis functions, reinforcement learning algorithms change the weights w in order to learn an approximate value function $\hat{V} = \Phi w$. The features Φ dictate what type of value functions an agent can represent. The choice of Φ strongly influences the agent's ability to compute a useful policy.

The majority of successful, large-scale RL applications involving function approximation required humans to design features. Manually constructing features is often a tedious trial-and-error process. Moreover, many hand-engineered features are tied to a specific domain and are therefore not useful for solving other problems. An agent must generate its own representations for it to be truly autonomous. There has been some recent research on algorithms for automatically generating features. In the rest of this section, we discuss features used in RL that are manually devised and those that are automatically generated.

2.4.1 Hand-coded Basis Functions

- Domain specific features

  There are many examples in the literature of researchers crafting hand-engineered features that are domain specific. These features are usually selected by the designer's intuition, an expert's knowledge of the domain, or simple trial-and-error. Three examples are:

  1. Tesauro's backgammon player, TD-Gammon [102]. After achieving moderate performance using just a raw state encoding, several expert features (e.g. strength of a blockade, probability of pieces being hit) were added to the state representation. These extra features boosted performance and ultimately resulted in a backgammon player that achieved world class play.

2. Crites and Barto's elevator dispatcher [23]. Forty-seven features were created for this domain that dealt with the status of the buttons in the elevator, the locations of the elevators relative to each other and to waiting passengers, et cetera.

3. Bertsekas and Tsitsiklis' Tetris player [12]. Twenty-two features were designed including information about the height of columns, the difference in adjacent column heights, and the number of holes. These features resulted in mediocre performance using a temporal difference learning learning algorithm [12]; however, the same 22 features were recently used in conjunction with the cross-entropy method (which is a type of Monte Carlo algorithm) improving performance by two orders of magnitude [100].

These examples are obviously domain specific and thus cannot be used for general MDPs.

- Polynomial functions

  In the context of Markov decision processes, polynomial function approximators take the form $\phi(s) = [1, s, s^2, \ldots, s^K]^T$ where $K \ll |S|$. Polynomials are *global* basis functions. Although these functions are computationally simple to generate, they do not efficiently represent smooth functions. These functions can also be numerically ill-conditioned depending on the values of s and K.

- Tile coding

  Tile coding is a type of local function approximator over partitions of the state space. A tiling is a partition of the state space. Each tiling contains many tiles (features) with the property that only one tile (within a tiling) is active for any given state. Since tiles are either active or inactive, the features are boolean-valued. The total number of active features is equal to the number of tilings. For example, consider a two dimensional state space (x, y) covering the square region $[0, 1]^2$. A tiling could

cover the region $[0, 1]^2$ and contain tiles with a rectangular shape (e.g. a tile could have 0.1 length in the x direction and 0.2 length in the y direction).

There are several considerations that can make tile coding design intensive: how many tilings to use, what are the shape of the tiles within a tiling, and how to choose the dimensions for the tiling in high dimensional problems. Another challenge is efficiently representing these functions, which is usually done using hashing.

- Radial basis functions

  Radial basis functions (RBFs) are real-valued features where the value depends on the distance to the RBF center. The feature value becomes larger as the distance to the center decreases. RBFs have the nice property that they vary smoothly and are differentiable. The simplest type of RBF takes the form $\phi(s) = \exp(-\frac{\|s-c\|^2}{2\sigma^2})$ where c is the center of the RBF and σ^2 is its variance. The variance dictates the support of the RBF (i.e. how far from the center c before $\phi(s)$ becomes negligible). There are also more complicated forms for RBFs where the distance metric can differ from the Euclidean metric and/or the support can be determined by a covariance matrix rather than a scalar variance σ^2.

  The RBF parameters are the number of basis functions, the location of the centers, the distance function, and the covariance matrix. Depending on the MDP, there can be significant engineering of the parameters to ensure the basis functions allow for appropriate generalization. That said, there has been recent work on automatically adapting RBF parameters using the cross-entropy method [70]. More information on RBFs and tile coding can be found in Sutton and Barto's textbook [98].

- Fourier basis functions

  Konidaris and Osentoski [53] evaluated the use of Fourier basis functions for reinforcement learning tasks with continuous state spaces. The Fourier basis consists of sines and cosines defined over the range of the state space variables. For example, in

one dimension, a K^{th} order Fourier basis function takes the form $\phi_i(s) = \cos(\pi i s)$ for $i = 0, \ldots, K$. This formulation is easily extended to multiple dimensions. For example, if the state space is three dimensional, then $\phi_i(s) = \cos(\pi c_i^T s)$ where $c_i = [c_{i,1}, c_{i,2}, c_{i,3}]$ with $c_{i,1}$, $c_{i,2}$, and $c_{i,3} \in [0, 1, \ldots, K]$. There are $(K+1)^3$ possible coefficient vectors c_i when the state space is three dimensional. Since the number of potential basis functions grows exponentially with the state space dimensionality, a method for selecting appropriate functions from this exponential set is required.

2.4.2 Automatically Learned Basis Functions

There have been several attempts to automatically learn basis functions for MDPs. Here we briefly review the relevant literature with an emphasis on the main ideas underlying these methods.

- Successor representations

  Dayan [27] proposed *successor representations* to approximate value functions. The main idea behind this work, as well as the majority of function approximators discussed in this section, is that representations should respect the MDP dynamics. The rationale is that, since a state is linked to its successor states by the Bellman equation, these states are similar and should allow for generalization. The successor representation essentially predicts future state occupancy by keeping track of observed transitions. This technique was designed for MDPs with discrete state spaces.

- Temporal neighborhoods

  Kretchmar and Anderson [55] proposed creating basis functions based on temporal neighborhoods. This work can be seen as an extension of successor representations to problems with continuous state. To do so, they use tilings over the state space and then monitor the transitions between tiles. This can work well on problems with low state space dimensionality, but encounters difficulty in scaling to higher dimensions.

- Spatial neighborhoods

  Drummond [31] considered the problem of transfer learning: how should information about several tasks in one domain be used to accelerate learning in a new task. He used techniques from computer vision to detect nonlinearities in the state space. The nonlinearities allow for partitioning the state space, resulting in a set of features that can be reused for solving new tasks.

- Multiple value function decomposition

  In the same vein as Drummond's work, Foster and Dayan [37] also studied how to efficiently reuse information when solving multiple tasks in a single domain. The goal of their work was to find common structure amongst a set of value functions. They used a probabilistic generative model of a value function using a mixture of Gaussians. The learned models allowed for decomposition of the state space.

- Manifold representations

  Smart proposed using manifolds to model the state space topology [91]. His rationale followed the same logic of Dayan [27] and Kretchmar and Anderson [55] that function approximation in MDPs must respect this topology. Specifically, the state space is partitioned into charts, where each chart has an embedding function that provides a basis for representing a value function. There are many questions left unaddressed in this work: how to allocate charts, the size of charts, the type of embedding functions to use, et cetera. However, the idea of using manifolds allows for a rigorous framework that previous approaches lacked.

- Bellman error basis functions

  The Bellman error, which is the difference between the current estimate of the value function and the "backed up" value function, can be used to automatically construct basis functions. This is useful because the Bellman error points in the direction of

the target value function. Recently, Keller et al. [50] proposed iteratively adding basis functions that are tuned to estimates of the Bellman error. Specifically, the Bellman error was mapped to a low dimensional space (with the constraint that states with similar Bellman errors should be mapped close together), and then states were aggregated in the low dimensional space to form new basis functions. Their work builds on ideas proposed by Bertsekas and Castañon [11], who were interested in accelerating the value iteration algorithm by exploiting information in the Bellman error.

Parr et al. [81] considered Bellman error basis functions theoretically without the complications of approximations and sampling. They proved that iteratively adding basis functions tuned to the exact Bellman error improves the bound on the distance from the optimal value function provided the MDP model parameters are known. If the model parameters are unknown, they described necessary conditions where the basis functions derived from samples would still improve the bound.

Petrik [83] used the MDP parameters P^π and R^π to generate basis functions using Krylov methods. This technique generates bases by multiplying the reward function by powers of the transition matrix: $[R^\pi, P^\pi R^\pi, (P^\pi)^2 R^\pi, (P^\pi)^3 R^\pi, \dots]$. This method results in basis functions that span the same subspace as the Bellman error basis functions considered by Parr et al. [81].

- Graph-based basis functions

  The following three methods generate basis functions from graphs. Each method models the state space topology using a graph. An undirected graph $G = (V, E, W)$ is formed where V is the set of vertices (representing MDP states), E is the set of edges that capture a local neighborhood relationship, and W is a matrix of edge weights. Please note that we use W to represent a graph's weight matrix and w to represent weights in the value function approximation $\hat{V} = \Phi w$.

We provide a brief sketch here of proto-value functions, diffusion wavelets, and geodesic Gaussian kernels. A more detailed description of the first two techniques is provided in Chapter 3.

1. Proto-value functions

   In a similar fashion to the manifold representation [91] described above, Mahadevan [64] proposed modeling the state space topology using a graph (instead of charts). Basis functions are automatically generated by computing eigenfunctions of a Laplace operator on the graph. For example, the combinatorial graph Laplacian [20] is defined as $L = D - W$ where D is a diagonal matrix of the row sums of weight matrix W (i.e. $D(i,i) = \sum_j W(i,j)$). Spectral analysis of the Laplacian operator finds the matrix's eigenvectors and eigenvalues: $L\phi_i = \lambda_i \phi_i$. These eigenvectors, or proto-value functions (PVFs), have global support on the graph and are ordered by their smoothness in the same spirit as the smooth trigonometric functions in Fourier analysis.

2. Diffusion wavelets

   Diffusion wavelets [22, 63] are a generalization of wavelets [25] to graphs. They are multiscale basis functions derived from a graph operator, such as the stochastic matrix $D^{-1}W$. The multiple scales come from representing different powers of the diffusion operator; smaller powers produce functions with more localized support on the graph while larger powers produce functions with more global support.

   A well known issue with global basis functions like PVFs is that they can have difficulty representing functions with discontinuities and/or different degrees of smoothness. The multiscale nature of diffusion wavelets allows for a more efficient representation of functions with discontinuities.

3. Geodesic Gaussian kernels

   Sugiyama et al. [94] also used a graph defined over the MDP state space. They proposed placing radial basis functions on the graph. These RBFs were termed geodesic Gaussian kernels (GGKs). The distance between the center of a GGK and another vertex in the graph is computed using Dijkstra's shortest path algorithm. It is important to point out that the shortest paths computed using Dijkstra's algorithm can produce unexpected results if there exist *shortcut* edges, whereas just using local distances (as used by PVFs and diffusion wavelets) tends to be more robust to shortcut edges.

2.5 Least-Squares Learning Algorithms

Reinforcement learning algorithms can differ in how samples from an environment are used to learn a value function. Online RL algorithms use each sample to directly update an estimated value function and then discard the sample. In contrast, least-squares RL algorithms [17, 16, 56] store statistics that capture information about the MDP. Each domain sample is used to update the statistics. When the value function or action-value function is needed, the statistics are used to generate an estimate. Least-squares algorithms make more efficient use of MDP samples and eliminate the need to tune the step-size parameter α (α was described in the TD(0) and Q-learning update equations in Section 2.2). The experiments in this dissertation are conducted using least-squares algorithms because of these benefits.

There are two common least-squares algorithms for RL: the fixed point (FP) method [17, 16] and the Bellman residual (BR) minimization method. These algorithms differ in the objective function that the least-squares method minimizes. The objective functions for both BR and FP involve functions of the Bellman residual $(T^{\pi}(\hat{V}) - \hat{V})$. We explain these algorithms and their differences in further detail in Chapter 4.

CHAPTER 3

GRAPH-BASED BASIS FUNCTIONS

In Section 2.4, we briefly introduced techniques that generate basis functions from graphs which represent a MDP state space. Here we provide a detailed description of two of these basis construction algorithms: proto-value functions and diffusion wavelets.

3.1 From MDPs to Graphs

The first step to generating graph-based basis functions is to form a graph from MDP samples. We focus here on weighted, undirected[1] graphs $G = (V, E, W)$ where V is a set of vertices, E is a set of edges, and W is a $|V| \times |V|$ weight matrix with $W(u, v) > 0$ if $(u, v) \in E$. If $(u, v) \notin E$, then $W(u, v) = 0$. The main idea is that the graph represents the topology of the state space. The vertices in the graph correspond to states in the MDP.[2] Edges are inserted between a pair of vertices depending on a user specified distance function (although the distance function itself can be learned from the MDP dynamics). A valuable aspect of the graph-based framework is that it works equally well in both discrete and continuous state spaces. For example, with a discrete state space, an edge can be placed between two vertices/states if one state can transition to the other. More general distance functions can also be used with discrete states. For continuous state spaces, there are two common approaches to building a graph from a set of sampled MDP states. Given an

[1] It is also possible to defined *directed* graphs from MDPs. We discuss the use of directed graphs at the end of Section 3.2.

[2] Osentoski [79, 78] has explored graphs representing the MDP state-action space topology. Basis functions from such graphs can be directly used to approximate action-value functions.

appropriate distance function comparing two states (e.g. Euclidean distance), each sampled state can be connected to its k nearest neighboring states or it can be connected to every sampled state within a specified minimum distance. Edge weights can either be set to a default value (e.g. 1) or can be set based on the distance metric. For a Euclidean distance metric $d(s_i, s_j) = \|s_i - s_j\|^2$, it is common to set the weight as $W(i,j) = \exp(\frac{-d(s_i,s_j)}{\sigma})$ for some $\sigma > 0$.

As a running example throughout this chapter, we use a grid MDP with 446 discrete states and the mountain car task which is continuous and two dimensional. All of the domains used in this dissertation are described in Appendix A. The graph for the discrete MDP simply contains edges between states adjacent via any of the four canonical actions. We use a k nearest neighbor graph for the mountain car MDP based on a weighted Euclidean distance metric. Figure 3.1 shows the structure of the discrete MDP and the graph used for the mountain car domain. For the mountain car plot, the black circles are the graph's vertices (sampled states) and the blue lines are the edges.

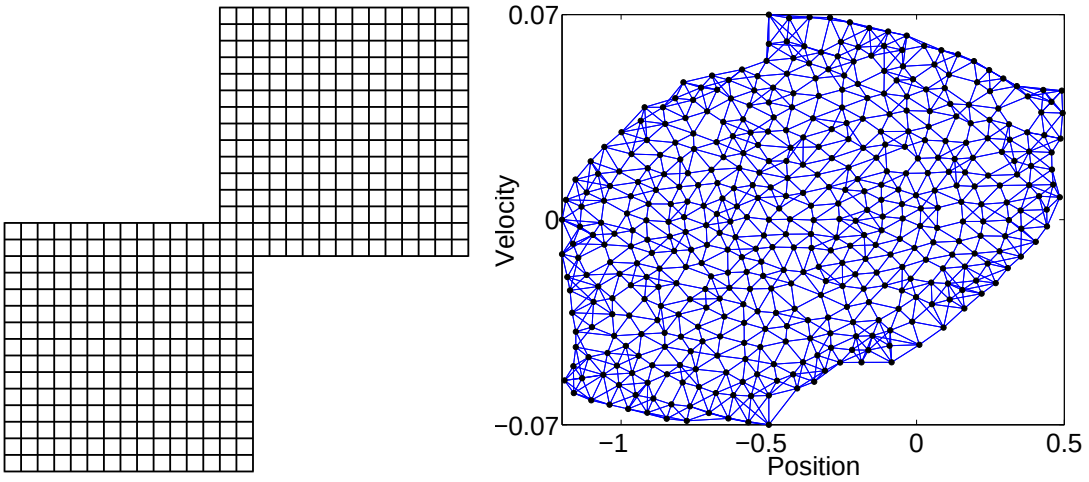


Figure 3.1. The discrete grid MDP (left) and graph constructed from the mountain car MDP (right) are running examples throughout Chapter 3.

3.2 Graph Laplacian

We make the simplifying assumption that the graph $G = (V, E, W)$ formed from the MDP samples is undirected, weighted, and connected. These are not restrictive assumptions. If the graph is disconnected, then each component can be considered individually. Directed graphs can also be used which we describe at the end of this section. The combinatorial graph Laplacian [20] is defined as $L = D - W$ where D is a diagonal matrix of the row sums of W (i.e. $D(i,i) = \sum_j W(i,j)$). The graph Laplacian has proven extremely useful in machine learning. The structure of the data encoded in the graph has been exploited for nonlinear dimensionality reduction [86, 101, 5], clustering [76], and semi-supervised learning [7]. The theoretical underpinning of the graph Laplacian is based on its convergence to the Laplace-Beltrami operator on the underlying manifold from which the data samples (graph vertices) are drawn [41, 6].

The combinatorial Laplacian L is symmetric and positive semidefinite; therefore, L has all real and non-negative eigenvalues. The Laplacian acts as an operator for functions on the graph. Given a function $f \in \mathbb{R}^{|V|}$ which has a value at each graph vertex, the multiplication Lf outputs a new function. The i$^{\text{th}}$ value of the vector Lf, which we abbreviate as $(Lf)_i$, is equal to $\sum_{j \sim i} W(i,j)(f(i) - f(j))$ where $j \sim i$ indicates an edge between vertices i and j. Thus, the Laplacian acts as a difference operator. This is useful in determining how smooth a function is according to the graph topology. The Dirichlet sum is defined as:

$$\langle f, Lf \rangle = f^T L f = \sum_{i \sim j} W(i,j) \left(f(i) - f(j)\right)^2 . \tag{3.1}$$

Notice the squared differences $(f(i) - f(j))^2$ are weighted by the strength of the connection. If the Dirichlet sum is small, then we know the function f is relatively smooth on the graph (i.e. neighboring vertices have similar values in the function f). We show in Chapter 4 how the ability to measure the smoothness of a function can be put to use as a regularization tool.

The normalized Laplacian [20] is defined as $\mathcal{L} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$. The normalized Laplacian, which is also a difference operator, accounts for the degree of each vertex individually. This can be seen by considering $(\mathcal{L}f)_i = \frac{1}{\sqrt{D(i,i)}} \sum_{j \sim i} W(i,j) \left(\frac{f(i)}{\sqrt{D(i,i)}} - \frac{f(j)}{\sqrt{D(j,j)}} \right)$. The normalized Laplacian is a symmetric matrix. It is related to the (possibly nonsymmetric) stochastic matrix $\mathcal{P} = D^{-1}W$, which can be though of as a random walk matrix, by the equation $\mathcal{P} = D^{-\frac{1}{2}}(I - \mathcal{L})D^{\frac{1}{2}}$.[3] This ensures that $\mathcal{L}$ and $\mathcal{P}$ have similar spectral properties. If $\mathcal{P}$ has an eigenvalue λ with associated eigenvector ϕ, then $\mathcal{L}$ has an eigenvalue $(1 - \lambda)$ with associated eigenvector $D^{-\frac{1}{2}}\phi$.

Mahadevan defined the term proto-value functions [64, 67] to be the eigenvectors of the graph Laplacian. The eigendecomposition of the combinatorial Laplacian is written $L\phi_i = \lambda_i \phi_i$ for $i = 1, \ldots, |V|$ or, in matrix format, $L\Phi = \Phi\Lambda$. The matrix $\Phi = \left[\phi_1, \phi_2, \ldots, \phi_{|V|}\right]$ contains (orthogonal) eigenvectors and Λ is a diagonal matrix with $\Lambda(i,i) = \lambda_i$. Note that all of the eigenvectors and eigenvalues are real-valued. The matrix Φ is a complete basis in that it can be used to represent any function on the graph. Let the numbering of the eigenvectors and eigenvalues be in terms of increasing eigenvalue. Given the properties of the Laplacian, we know $0 = \lambda_1 < \lambda_2 \leq \lambda_3 \leq \ldots \leq \lambda_{|V|}$. Furthermore, ϕ_1 is a constant vector. The increasing order of eigenvalues leads to an ordering of the eigenvectors by "smoothness." We can see this by considering the Dirichlet sum of an eigenvector ϕ_i: $\phi_i^T L \phi_i = \phi_i^T \lambda_i \phi_i = \lambda_i$. To visualize the idea of smooth eigenvectors, Figure 3.2 shows the second through fourth eigenvectors of the combinatorial graph Laplacian for the discrete grid MDP and the mountain car task (the first eigenvector is omitted since it is just a constant function). Mahadevan and Maggioni proposed using the smoothest K eigenvectors as a basis for representing value functions and action-value functions [67]. If value functions are smooth with respect to the graph topology, then this basis will be useful

[3] To clarify, we use the notation P to refer to a MDP transition function, P^π to refer to a probability transition matrix associated with policy π, and $\mathcal{P} = D^{-1}W$ to refer to a random walk matrix on a graph.

for approximation. The construction of the basis functions was one step in an overall approximate policy iteration algorithm which we describe in Section 4.5.2.

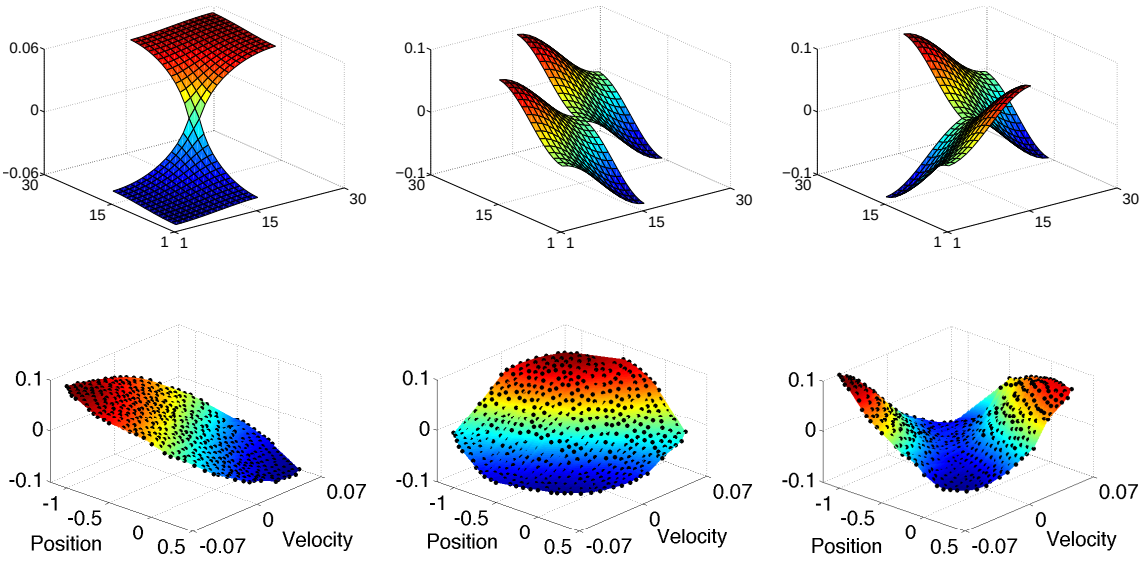


Figure 3.2. Second to fourth Laplacian eigenvectors for the discrete grid MDP (top row) and the mountain car MDP (bottom row).

Note that the graph Laplacian was defined using sampled states of the MDP. Therefore, the Laplacian eigenvectors are features $\phi(s)$ defined over the MDP state space. It is straightforward to use these features for approximating value functions: $\hat{V}(s) = \phi(s)^T w$. For MDPs with discrete actions, the same set of basis functions can also be used to approximate the action-value function $\hat{Q}(s,a) = \phi(s,a)^T w$. The idea is to use the same set of features for each action separately by padding the features with zeros. For example, consider a MDP with two actions, a_1 and a_2. The approximate action-value function can take the form:

$$\hat{Q} = \begin{bmatrix} \hat{Q}(\cdot, a_1) \\ \hat{Q}(\cdot, a_2) \end{bmatrix} = \begin{bmatrix} \Phi & \mathbf{0} \\ \mathbf{0} & \Phi \end{bmatrix} \begin{bmatrix} w_{a_1} \\ w_{a_2} \end{bmatrix}.$$

If Φ has K columns, then the feature vector $\phi(s,a)$ has $K \cdot |A|$ values of which only K are (potentially) nonzero.

To be complete, we point out two generalizations of the undirected graph Laplacian. The Laplacian is well-defined for directed graphs [21]. Given a weight matrix W_d and diagonal row sum matrix D_d associated with a directed graph, $\mathcal{P} = D_d^{-1} W_d$ is a stochastic matrix. We denote the dominant left eigenvector of $\mathcal{P}$ as ψ (i.e. $\psi^T \mathcal{P} = \psi^T$), which is also known as the Perron vector. If the directed graph is strongly connected and aperiodic (properties which can be guaranteed using a technique known as the teleporting random walk [80]), then the Perron-Frobenius theorem ensures ψ is unique and contains only positive real values. The vector ψ is the invariant distribution upon convergence of the random walk. The combinatorial and normalized Laplacians are defined respectively as:

$$L_d = \Psi - \frac{\Psi\mathcal{P} + \mathcal{P}^T\Psi}{2}$$
$$\mathcal{L}_d = I - \frac{\Psi^{1/2}\mathcal{P}\Psi^{-1/2} + \Psi^{-1/2}\mathcal{P}^T\Psi^{1/2}}{2},$$

where Ψ is a diagonal matrix with $\Psi(i,i) = \psi(i)$. In previous work [44], we compared using directed and undirected Laplacian eigenvectors as basis functions for RL tasks. Both directed Laplacians, though starting from a nonsymmetric matrix $\mathcal{P}$, are in fact symmetric matrices. Essentially, the matrices are made symmetric by the Perron vector. There is also a notion of nonsymmetric Laplacian matrices [1]. These are matrices with (1) non-positive off-diagonal elements, and (2) zero row sums. There is a direct connection between these matrices and MDPs [65]. Lastly, we note there is a connection between the graph Laplacian and reproducing kernels [92, 65]. Specifically, the pseudoinverse of the Laplacian is a reproducing kernel. The field of spectral graph theory studies other properties associated with the Laplacian and its spectra [20].

3.3 Diffusion Wavelets

Diffusion wavelets [22, 63] are a generalization of classical wavelets [25] to graphs and manifolds. They are a representation of powers of a diffusion process on a graph. At small

powers, the diffusion process has local effects while at larger powers the diffusion has more global effects. By representing multiple powers of a diffusion process, diffusion wavelets allow for a multiscale analysis of graphs.

An intuitive way to understand diffusion wavelets is to compare and contrast them with graph Laplacian eigenvectors. The similarity of Laplacian eigenvectors and diffusion wavelets is that they are both basis functions generated from a graph. Both sets of basis functions are adapted to the topology of the graph (MDP state space) and can be used as a basis for approximating functions on the graph. Laplacian eigenvectors have global support on the graph. In other words, each eigenvector takes on a value at every vertex in the graph. Diffusion wavelets, on the other hand, are multiscale basis functions. By multiscale, we mean they have varying degrees of support on the graph. The functions range from having very localized support (i.e. only a few vertices on the graph take on a value) all the way up to global support. There are $|V|$ Laplacian eigenvectors which provide a complete basis for functions on the graph. There are more than $|V|$ diffusion wavelet basis functions which means they provide an overcomplete basis. The extra flexibility of an overcomplete, multiscale basis means diffusion wavelets can represent certain functions more efficiently than Laplacian eigenvectors. It is well known from Fourier analysis that global functions like the Laplacian eigenvectors can have difficulty representing functions with discontinuities and/or varying degrees of smoothness in different regions. This issue in fact prompted the construction of wavelets.

Before describing the diffusion wavelet construction, we present a few examples for the discrete grid MDP and the mountain car MDP. Figure 3.3 shows three diffusion wavelet scaling functions for the two domains. The three functions for both domains consist of one function with local support, one function with an intermediate level of support, and one function with global support. Notice in particular that the functions with global support look very similar to the eigenvectors shown in Figure 3.2.

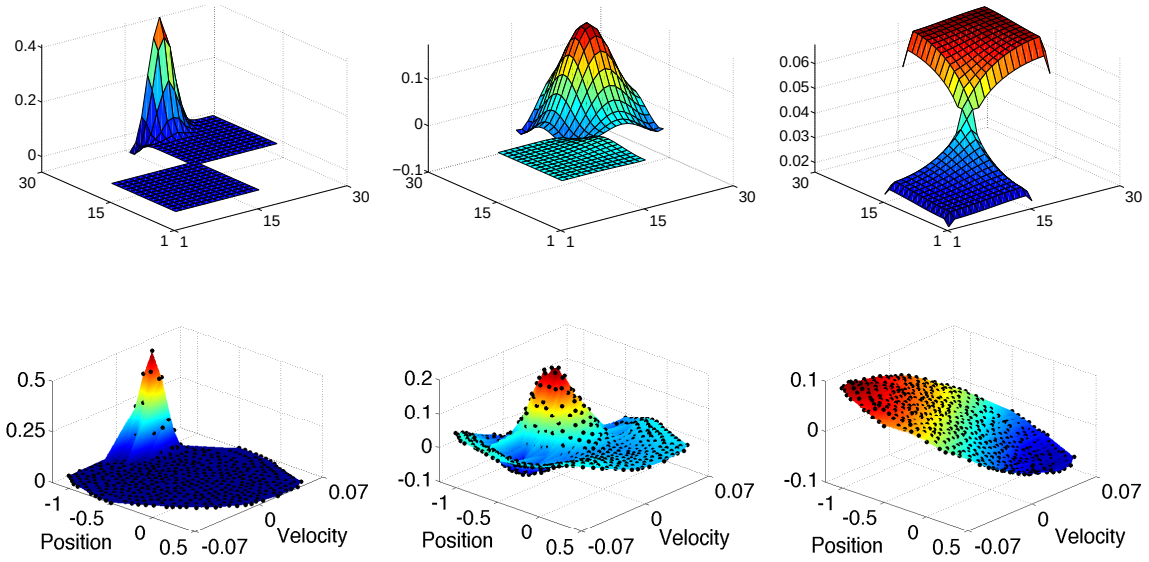


Figure 3.3. Functions (from local to global) in the diffusion wavelet tree for the discrete grid MDP (top row) and the mountain car MDP (bottom row).

Figure 3.4 gives a high level description of the diffusion wavelet algorithm. The QR decomposition of a matrix is needed in the construction. Given a matrix A, its QR decomposition is written $A = QR$ where Q is an orthogonal matrix and R is an upper triangular matrix.

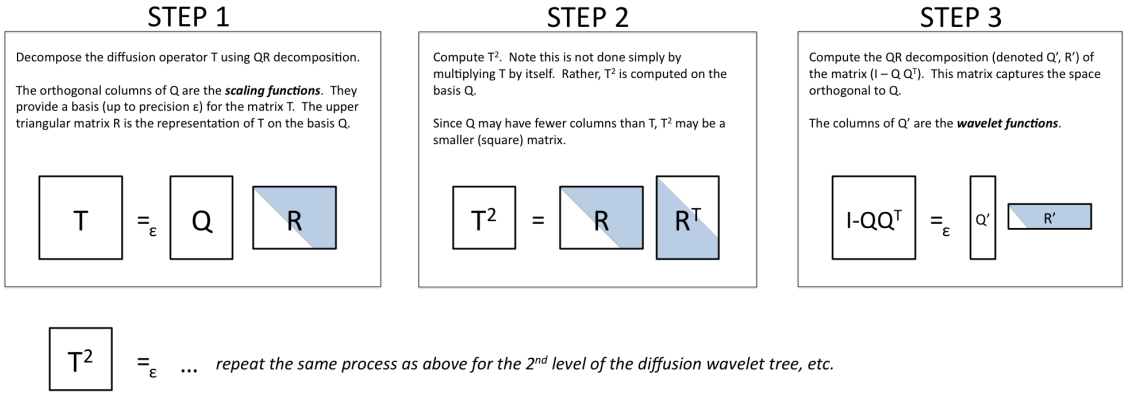


Figure 3.4. High level description of the diffusion wavelet tree construction. The square and rectangular boxes represent matrices. The shading of matrices R and R' indicate the location of nonzero values.

The diffusion wavelet construction begins with the definition of the diffusion operator as $T = (I - \mathcal{L})$ with powers T^t, $t > 0$. To make the diffusion aspect more obvious, this can be rewritten $T = D^{-0.5}WD^{-0.5} = D^{0.5}\mathcal{P}D^{-0.5}$ where $\mathcal{P} = D^{-1}W$ is a stochastic matrix representing a random walk (diffusion process) on the graph. Note that $\mathcal{P}$ is *conjugate* along with its powers to T; thus, studying T and $\mathcal{P}$ are equivalent in terms of spectral properties. It is computationally easier to deal with T since it is symmetric. Small powers of T^t correspond to short-term behavior in the diffusion process and large powers correspond to long-term behavior. Diffusion wavelets are naturally multiscale basis functions because they account for increasing powers of T^t (in particular, the dyadic powers $t = 2^j$). We give a brief sketch of the diffusion wavelet algorithm; a more thorough description can be found in the original paper [22]. Aside from matrix T, the other inputs to the algorithm are J, the maximum number of levels to compute, ϵ, a precision parameter, and $\mathsf{SpQR}(A, \epsilon)$, a sparse QR algorithm that outputs (sparse) matrices Q and R such that $A =_\epsilon QR$ (i.e. the columns of Q ϵ-span the columns of A). The outputs of the algorithm are a set of scaling functions $\{\phi_j\}$ and wavelet functions $\{\psi_j\}$ at different levels/scales. As the level j gets larger, the number of scaling and wavelet functions gets smaller because the diffusion process spreads out and becomes more compressible. Algorithm 1 shows the details of the construction and uses the following notation: $[T]_{\phi_a}^{\phi_b}$ is a matrix representing T with respect to the basis ϕ_a in the domain and ϕ_b in the range ($n_b \times n_a$ matrix) and $[\phi_b]_{\phi_a}$ is a set of functions ϕ_b represented on the basis ϕ_a ($n_a \times n_b$ matrix). Typically the initial basis for the algorithm is ϕ_0 is assumed to be the delta functions, but this is not strictly necessary.

The diffusion wavelet construction proceeds by computing the sparse QR decomposition (up to a precision of ϵ) of $[T]_{\phi_0}^{\phi_0}$. This provides (1) a new basis $[\phi_1]_{\phi_0}$ which is defined in the range of the old basis ϕ_0, and (2) a representation of the diffusion operator $[T]_{\phi_1}^{\phi_0}$ defined in the range of the new basis ϕ_1. The second power of the diffusion operator (defined completely in the new basis ϕ_1) is computed as $[T^2]_{\phi_1}^{\phi_1} = [T]_{\phi_1}^{\phi_0}([T]_{\phi_1}^{\phi_0})^*$ where the symbol * indicates the conjugate transpose. Notice that the size of matrix T^2 may be smaller than the

Algorithm 1: `Diffusion Wavelet Tree`

Input: $[T]_{\phi_0}^{\phi_0}$, diffusion operator in the basis ϕ_0
ϕ_0, initial basis for T (usually unit basis vectors)
J, maximum number of levels in tree
`SpQR`, a sparse QR algorithm with two inputs: a matrix and parameter ϵ
$\epsilon > 0$, determines precision level of QR decomposition

Output: $\{\phi_j\}_{j=1}^{J}$, scaling functions by level
$\{\psi_j\}_{j=0}^{J-1}$, wavelet functions by level

for $j = 0$ to $(J-1)$ **do**
$[\phi_{j+1}]_{\phi_j}$, $[T^{2^j}]_{\phi_j}^{\phi_{j+1}} \leftarrow \mathtt{SpQR}([T^{2^j}]_{\phi_j}^{\phi_j}, \epsilon)$
$[T^{2^{j+1}}]_{\phi_{j+1}}^{\phi_{j+1}} \leftarrow [T^{2^j}]_{\phi_j}^{\phi_{j+1}}([T^{2^j}]_{\phi_j}^{\phi_{j+1}})^*$
$[\psi_j]_{\phi_j} \leftarrow \mathtt{SpQR}(I_{\langle\phi_j\rangle} - [\phi_{j+1}]_{\phi_j}([\phi_{j+1}]_{\phi_j})^*, \epsilon)$
end for

size of T. The last step is to compute the wavelet functions $[\psi_0]_{\phi_0}$. This is accomplished by using the sparse QR algorithm on the matrix $\left(I - [\phi_1]_{\phi_0}[\phi_1]_{\phi_0}^*\right)$. Notice that the span of the scaling functions $[\psi_0]_{\phi_0}$ is the orthogonal complement of the span of $[\phi_0]$ into $[\phi_1]$. In other words, the wavelet functions at level $j = 0$ capture the detail lost in going from basis $[\phi_0]$ to the new basis $[\phi_1]$ (i.e. the wavelet functions act as a high-pass filter). It is also possible to further decompose the wavelet functions into wavelet packets [18]. This procedure then proceeds iteratively until the maximum level J is reached or until the number of scaling functions goes beneath a minimum threshold. Note that the scaling functions $[\phi_j]_{\phi_{j-1}}$ provide a mapping from level $j-1$ to level j. In order to view the functions in the original basis ϕ_0 (which is usually assumed to be the unit basis), the mapping is unrolled to give $[\phi_j]_{\phi_0} = [\phi_j]_{\phi_{j-1}}[\phi_{j-1}]_{\phi_{j-2}} \cdots [\phi_1]_{\phi_0}[\phi_0]_{\phi_0}$.

Maggioni and Mahadevan [63] proposed using diffusion wavelets for value function approximation. They used a heuristic for selecting a set of functions from the diffusion wavelet tree. Given a desired number of basis functions K, they selected the scaling functions at level J, then the wavelet functions at level $J-1$, then the wavelet functions at level $J-2$, etc. until K functions are selected. This heuristic generates an orthogonal basis consisting of the most global functions in the diffusion process. This is very similar in spirit to selecting the K "smoothest" Laplacian eigenvectors as a basis. Both these proce-

dures are independent of the value function being estimated. This can be an inefficient use of the diffusion wavelet and Laplacian eigenvector dictionaries. In Chapter 6, we explore algorithms for selecting basis functions from both dictionaries based on a specific policy's value function.

CHAPTER 4

HYBRID LEAST-SQUARES METHODS AND GRAPH-BASED REGULARIZATION

In this chapter, we develop hybrid least-squares algorithms for approximate policy evaluation [48]. Least-squares methods are important because they are sample efficient and do not require tuning a step-size parameter. The hybrid algorithms are a parametric combination of two common least-squares RL methods. When used within policy iteration, we show the use of hybrid algorithms can, in some instances, lead to better policies. We also describe how the graph Laplacian can be used to provide regularization and prevent overfitting from noisy samples.

4.1 Previous Work

Least-squares reinforcement learning algorithms [17, 16, 56] store statistics from MDP samples. The Bellman residual (BR) minimization method and the fixed point (FP) method both store a matrix $A \in \mathbb{R}^{K \times K}$ and a vector $b \in \mathbb{R}^K$ where K is the number of basis functions. Informally, the matrix A captures information about state transitions and the vector b stores information about the reward function. When the approximate value function is needed, the least-squares problems $Aw = b$ is solved to give $\hat{V} = \Phi w$.

Below we describe the BR and FP methods as well as two other approximate policy evaluation techniques.[1]

[1]The first three techniques were similarly described by Munos [74].

4.1.1 Optimal Approximate Method

If the target value function V^π were known, then it is easy to find an approximation $\hat{V}$ simply by projecting V^π onto the space spanned by the basis functions. This directly minimizes the loss function $L_{OPT}(w) = \|\hat{V} - V^\pi\|_\rho$, where the errors for each state are weighted according to distribution ρ. Thus, the solution is $\hat{V} = \Phi w = \Pi_\rho V^\pi$ where $\Pi_\rho = \Phi(\Phi^T D_\rho \Phi)^{-1}\Phi^T D_\rho$ is a projection matrix and D_ρ is a diagonal matrix $D_\rho(i,i) = \rho(i)$. The difficulty of this method is in computing V^π, which can in principle be done using Monte Carlo methods.

4.1.2 Fixed Point Method

This technique, originally proposed by Bradtke and Barto [17] and later generalized by Boyan [16], computes a solution by forcing $\hat{V}$ to be a fixed point of the Bellman operator. Since the Bellman operator can back up values out of the space spanned by the basis functions, it must be followed by a projection onto the column space of Φ (written $[\Phi]$) to ensure $\hat{V}$ is a fixed point. Thus, the solution is to minimize the loss function $L_{FP}(w)$:

$$\begin{aligned}
\min_w \; L_{FP}(w) &= \min_w \; \|\Pi_\rho T^\pi(\hat{V}) - \hat{V}\|_\rho^2 \\
&= \min_w \; \|\Pi_\rho (T^\pi(\hat{V}) - \hat{V})\|_\rho^2 \\
&= \min_w \; \|\Pi_\rho (R^\pi + \gamma P^\pi \Phi w - \Phi w)\|_\rho^2. \qquad (4.1)
\end{aligned}$$

In the third line above, note that $\hat{V} = \Pi_\rho \hat{V}$. The least-squares solution to this problem is to find w such that $A_{FP} w = b_{FP}$ where:

$$\begin{aligned}
A_{FP} &= \Phi^T D_\rho (I - \gamma P^\pi)\Phi \\
b_{FP} &= \Phi^T D_\rho R^\pi.
\end{aligned}$$

We refer to this technique as the FP solution to be consistent with previous work [56], but it has also been referred to as least-squares temporal difference (LSTD) learning [17, 16]

and as the temporal difference method [88, 74]. Unbiased estimates of the matrix A_{FP} and vector b_{FP} can be obtained from a single sample $\langle s, \pi(s), r', s' \rangle$ by the following updates:

$$\hat{A}_{FP} = \hat{A}_{FP} + \rho(s)\phi(s)(\phi(s) - \gamma\phi(s'))^T$$
$$\hat{b}_{FP} = \hat{b}_{FP} + \rho(s)\phi(s)r'.$$

4.1.3 Bellman Residual Minimization

This technique computes a solution by minimizing the magnitude of the Bellman residual where the errors for each state are weighted according to distribution ρ. Thus, the solution is to minimize the loss function $L_{BR}(w)$:

$$\begin{aligned} \min_w \; L_{BR}(w) &= \min_w \; \|T^\pi(\hat{V}) - \hat{V}\|_\rho^2 \\ &= \min_w \; \|R^\pi + \gamma P^\pi \Phi w - \Phi w\|_\rho^2. \end{aligned} \tag{4.2}$$

The least-squares solution is to minimize $\|A_{BR}w - b_{BR}\|_\rho^2$ where:

$$A_{BR} = \Phi^T(I - \gamma P^\pi)^T D_\rho (I - \gamma P^\pi)\Phi$$
$$b_{BR} = \Phi^T(I - \gamma P^\pi)^T D_\rho R^\pi.$$

This technique, proposed by Schweitzer and Seidmann [89], has also been referred to as the residual-gradient method [3, 88], the quadratic residual method [74], and as the Bellman residual method [56, 2]. To achieve an *unbiased* estimate of A_{BR} and b_{BR}, two samples from each state are required (see Chapter 8.5 [98]). The feasibility of getting two samples depends on the application. Given double samples $\langle s, \pi(s), r', s' \rangle$ and $\langle s, \pi(s), r'', s'' \rangle$, the updates are

$$\hat{A}_{BR} = \hat{A}_{BR} + \rho(s)(\phi(s) - \gamma\phi(s'))(\phi(s) - \gamma\phi(s''))^T$$
$$\hat{b}_{BR} = \hat{b}_{BR} + \rho(s)(\phi(s) - \gamma\phi(s'))r'.$$

If only a single sample is available, then replacing $\phi(s'')$ with $\phi(s')$ in the equation above for $\hat{A}_{BR}$ results in a biased estimate of A_{BR}. This occurs because the term $\gamma^2\Phi^T(P^\pi)^T D_\rho P^\pi \Phi$ in A_{BR} cannot be estimated from just a single transition. Two common heuristics for dealing with this issue are to hypothesize a second sample using a nearest neighbor of s' and to not update $\hat{A}_{BR}$ until a state s has been visited at least twice. Recently, Antos et al. [2] proposed a technique for avoiding double samples by adding an auxiliary function which itself must be optimized.

4.1.4 Residual Algorithms

The BR solution minimizes the Bellman residual (Equation 4.2) and the FP solution minimizes the projected Bellman residual (Equation 4.1). Baird [3] proposed *residual algorithms* as a way to combine these techniques. The term "residual" algorithm was used to emphasize that it was different from a "residual-gradient" algorithm (his terminology for BR). To avoid any confusion, we refer to residual algorithms as *hybrid algorithms*. This name also emphasizes the fact that it is a combination of BR and FP. Baird's original version was an incremental algorithm. An update to the weight vector was computed by linearly combining the updates due to the BR and FP: $\Delta w_H = \xi \Delta w_{BR} + (1-\xi)\Delta w_{FP}$ where $\xi \in [0, 1]$. In the next section, we introduce two ways to formulate the hybrid technique using least-squares methods.

4.2 Hybrid Least-Squares Algorithms

The hybrid approach accounts for both the Bellman residual (which is minimized by the BR in Equation 4.2) *and* the projection of the Bellman residual onto $[\Phi]$ (which is minimized by the FP in Equation 4.1). In this section, we examine two ways of combining these objectives to derive hybrid least-squares algorithms H_1 and H_2. The difference between the H_1 and H_2 derivations is when the fixed point constraint (i.e. $\hat{V} = \Pi T^\pi(\hat{V})$) is enforced.[2]

[2]We thank Marek Petrik for help with the H_1 algorithm.

Both methods when used with an exact representation produce the target value function V^{π}. When using an approximate representation, they produce different results and have different storage and computational requirements. The majority of this section, Section 4.3, and the experiments in Section 4.6 appeared in our previous work [48].

4.2.1 Motivation

There are three factors that motivate hybrid algorithms. First, as pointed out by Baird [3], hybrid algorithms are a general class of algorithms that include the BR and FP algorithms as special cases at opposite ends of a spectrum. Fully understanding this spectrum is worthwhile in its own right. Also, as least-squares techniques have been applied to the BR and FP algorithms [17, 16, 56] to make them more data efficient than their incremental counterparts, it makes sense to design a least-squares version of hybrid algorithms. These least-squares algorithms have an intuitive geometric perspective. The BR and FP methods minimize the length of different sides of a triangle defined by the Bellman equation [56]. Hybrid algorithms naturally complete this perspective.

The second factor motivating hybrid least-squares algorithms is the empirical behavior of approximate policy iteration. The FP algorithm tends to produce better policies than the BR algorithm [56]. However, this increase in performance comes at the expense of stability [3, 74]. Li [61] analyzed incremental versions of the FP and BR methods under a particular learning model and concluded that the BR method can achieve smaller residuals while the FP method can make more accurate predictions. It is also worth noting that the Bellman residual is used to provide theoretical performance bounds [109, 74]. In other words, having a small magnitude of the Bellman residual (which the BR method focuses on) translates into guarantees on the quality of the approximate value functions. It is more difficult to provide such guarantees using the FP method [74]. Table 4.1 lists some properties associated with the BR and FP methods.

BR	FP
• Minimizes the norm of the Bellman residual which is directly related to performance bounds	• Minimizes the norm of the projected Bellman residual
• Least-squares solution is *biased* when only single-samples from MDP are available	• Least-squares solution is *unbiased* when single-samples from MDP are available
• Performs backward-bootstrapping (i.e. the value of a state is influenced by its predecessors as well as its successors)	• Does not perform backward-bootstrapping
• Empirically performs worse than FP when used in policy iteration	• When used in policy iteration, often finds much better policies than BR provided policy iteration converges

Table 4.1. Properties and behavior of two common RL least-squares algorithms: the Bellman residual (BR) minimization method and the fixed point (FP) method.

Hybrid algorithms have the potential to achieve both stability and improved performance. To illustrate this on a concrete example, consider the six state MDP shown in Figure 4.1 with discount factor $\gamma = 0.99$. The optimal policy is to move right in the first three states and left in the last three states ($\pi^* = RRRLLL$). Let the initial policy be $\pi_0 = LLLLLL$ and assume there are three basis functions corresponding to the first three eigenvectors of the graph Laplacian [64]. These basis functions are symmetric and expressive enough to represent an approximate value function whose corresponding greedy policy is π^*. The basis functions are shown in Figure 4.2. The distribution ρ can be set to either the invariant distribution of P^{π_0} or the uniform distribution (which is appropriate when performing policy iteration [51]); the results hold for both distributions. The approximate value functions $\hat{V}_{BR}^{\pi_0}$ and $\hat{V}_{FP}^{\pi_0}$ were computed according to the least-squares solutions described in Section 4.1. Then the model was used to determine a policy π_1 that is greedy with respect to $\hat{V}$. The BR method produces a policy $\pi_1 = LLLLLL$ while the FP method produces a policy $\pi_1 = RRRRRR$. Thus, after one round of policy iteration, the BR method converges on the initial policy and the FP method completely flips the policy. Moreover, since the model and basis functions are symmetric, the FP method oscillates forever between $LLLLLL$ and $RRRRRR$. This example demonstrates the stability of the BR method and the FP method's

potential instability. We will revisit this example later to show that hybrid least-squares algorithms find solutions between these two extremes.

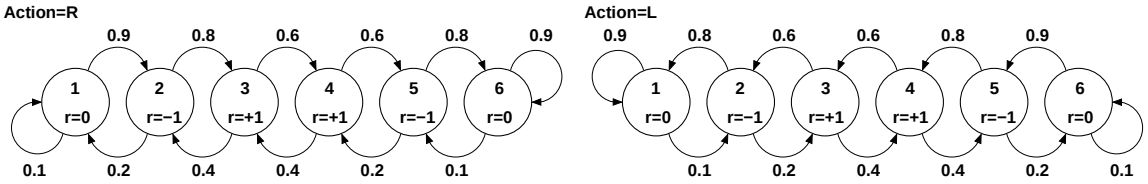


Figure 4.1. Reward and transition functions for a six state MDP with two possible actions.

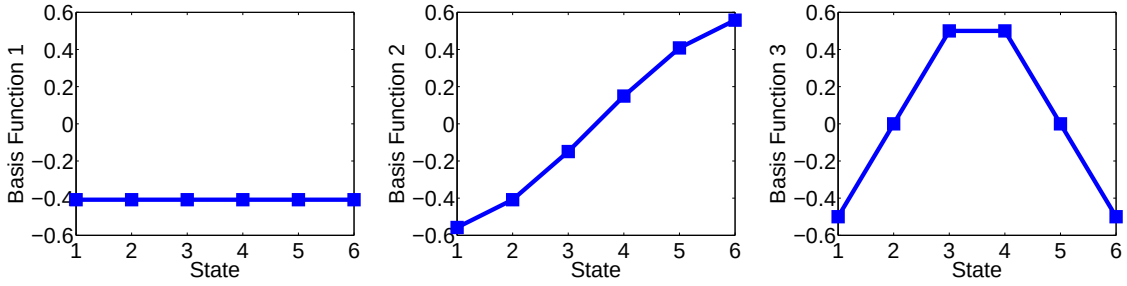


Figure 4.2. First three Laplacian eigenvectors associated with the MDP in Figure 4.1.

The third motivating factor is the bias associated with $\hat{A}_{BR}$ when only a single sample is available from each state. Denote the sampled Bellman update for a fixed policy π as $\hat{T}$. Note this is a random variable which depends on the samples. The expected value is $E[\hat{T}] = T^{\pi}$. Antos et al. [2] showed that the expected value of the estimated Bellman residual when using single samples is equal to the true Bellman residual plus the variance of the sampled Bellman update. This takes the form:

$$E\left[\|\hat{T}(\hat{V}) - \hat{V}\|_{\rho}^{2}\right] = \|T^{\pi}(\hat{V}) - \hat{V}\|_{\rho}^{2} + \rho^{T} Var\left[\hat{T}(\hat{V})\right],$$

where the variance $Var[\cdot]$ is point-wise and ρ is a distribution vector. Practically, this means that in minimizing the estimated Bellman residual, function approximation resources are spent minimizing the variance of $\hat{T}(\hat{V})$. Clearly, this is undesirable. On the other hand, $\hat{A}_{FP}$

is unbiased when only a single sample is available from each state. Hybrid algorithms, by naturally combining the BR and FP methods, offer control over the impact of the bias.

4.2.2 Algorithm $\mathbf{H}_1$

We combine the BR minimization problem (Equation 4.2) and the FP minimization problem (Equation 4.1) with a parameter $\xi \in [0,1]$. The loss function is $L_{H_1}(w) = \xi L_{BR}(w) + (1-\xi) L_{FP}(w)$. Simply combining these two problems results in a minimization over two separate norms; however, these can be combined into a single norm as we prove below:

$$
\begin{aligned}
\min_w \; & [\xi L_{BR}(w) + (1-\xi) L_{FP}(w)] \\
&= \min_w \; \left[\xi \|T^\pi(\hat{V}) - \hat{V}\|_\rho^2 + (1-\xi)\|\Pi_\rho(T^\pi(\hat{V}) - \hat{V})\|_\rho^2\right] \\
&= \min_w \; \left[\xi \|(I - \Pi_\rho + \Pi_\rho)(T^\pi(\hat{V}) - \hat{V})\|_\rho^2 + (1-\xi)\|\Pi_\rho(T^\pi(\hat{V}) - \hat{V})\|_\rho^2\right] \\
&= \min_w \; \left[\xi \|(I - \Pi_\rho)(T^\pi(\hat{V}) - \hat{V})\|_\rho^2 + \|\Pi_\rho(T^\pi(\hat{V}) - \hat{V})\|_\rho^2\right] \\
&= \min_w \; \|\sqrt{\xi}(I - \Pi_\rho)(T^\pi(\hat{V}) - \hat{V}) + \Pi_\rho(T^\pi(\hat{V}) - \hat{V})\|_\rho^2 \\
&= \min_w \; \|(\sqrt{\xi} I + (1 - \sqrt{\xi})\Pi_\rho)(T^\pi(\hat{V}) - \hat{V})\|_\rho^2 \\
&= \min_w \; \|(\sqrt{\xi} I + (1 - \sqrt{\xi})\Pi_\rho)(R^\pi + \gamma P^\pi \Phi w - \Phi w)\|_\rho^2. \qquad (4.3)
\end{aligned}
$$

The chain of steps relies on the Pythagorean theorem (used in both the fourth and fifth lines) and the fact that $[I - \Pi_\rho]$ and $[\Pi_\rho]$ are orthogonal subspaces. A least-squares equation of the form $\|A_{H_1} w - b_{H_1}\|_\rho^2$ can be derived[3] from the minimization problem in Equation 4.3:

$$
\begin{aligned}
A_{H_1} &= \Phi^T (I - \gamma P^\pi)^T D_\rho (\xi I + (1-\xi)\Pi_\rho)(I - \gamma P^\pi)\Phi \\
b_{H_1} &= \Phi^T (I - \gamma P^\pi)^T D_\rho (\xi I + (1-\xi)\Pi_\rho) R^\pi.
\end{aligned}
$$

[3] In expanding Equation 4.3 to form A_{H_1} and b_{H_1}, we use the following simplification: $\left(\sqrt{\xi} I + (1 - \sqrt{\xi})\Pi_\rho\right)^T D_\rho \left(\sqrt{\xi} I + (1 - \sqrt{\xi})\Pi_\rho\right) = D_\rho \left(\xi I + (1-\xi)\Pi_\rho\right)$.

To estimate A_{H_1} and b_{H_1} from samples, it is necessary to store three $K \times K$ matrices and two length K vectors. This can be seen by rewriting the equations:

$$A_{H_1} = \xi A_{BR} + (1-\xi) A_{FP}^T C^{-1} A_{FP}$$
$$b_{H_1} = \xi b_{BR} + (1-\xi) A_{FP}^T C^{-1} b_{FP},$$

where $C = \Phi^T D_\rho \Phi$. Thus, A_{H_1} can be estimated by incrementally updating $\hat{A}_{BR}$, $\hat{A}_{FP}$, $\hat{b}_{BR}$, $\hat{b}_{FP}$, as well as the matrix $\hat{C}$ via $\hat{C} = \hat{C} + \rho(s)\phi(s)\phi(s)^T$ given sample $\langle s, \pi(s), r', s' \rangle$. If only a single sample is available from each state, then $\hat{A}_{H_1}$ will be a biased estimate of A_{H_1} because of the bias in $\hat{A}_{BR}$. However, as mentioned in Section 4.2.1, hybrid algorithms can reduce the impact of the bias. This is achieved simply by setting ξ to a value less than one, thereby reducing the bias of the term $E\left[\xi \|\hat{T}(\hat{V}) - \hat{V}\|_\rho^2\right]$.

Both the BR and FP least-squares problems only need to store one $K \times K$ matrix and one length K vector, whereas the H_1 method requires three matrices and two vectors. Moreover, the matrix C must be inverted when computing A_{H_1}. These issues motivated our second implementation.

4.2.3 Algorithm $\mathbf{H}_2$

The algorithm H_1 was derived by linearly combining the loss functions $L_{BR}(w)$ and $L_{FP}(w)$. The loss function $L_{FP}(w)$, by virtue of using the projection matrix Π_ρ, directly enforces the fixed point constraint. An alternative option is to enforce the fixed point constraint *after* finding a solution to the minimization problem. More specifically, consider the following combined loss function:

$$L_{H_2}(w) = \left[\frac{\xi}{2} L_{BR}(w) + \frac{1-\xi}{2} L_{FP}(u, w)\right],$$

where $L_{FP}(u, w) = \|T^\pi(\Phi u) - \Phi w\|_\rho^2$. We first find the coefficient vector w that minimizes $L_{H_2}(w)$ for an arbitrary vector u and then enforce the fixed point constraint $u = w$.

This technique was recently described by Kolter and Ng [52]. Differentiating $L_{H_2}(w)$ with respect to w, we obtain:

$$\begin{aligned}\frac{\partial L_{H_2}(w)}{\partial w} &= \xi \left[\frac{\partial}{\partial w}\left(T^\pi(\Phi w) - \Phi w\right)\right]^T D_\rho \left(T^\pi(\Phi w) - \Phi w\right) + \\ &\quad (1-\xi)\left[\frac{\partial}{\partial w}\left(T^\pi(\Phi u) - \Phi w\right)\right]^T D_\rho \left(T^\pi(\Phi u) - \Phi w\right) \\ &= \xi \left(\gamma P^\pi \Phi - \Phi\right)^T D_\rho \left(T^\pi(\Phi w) - \Phi w\right) + (1-\xi)\left(-\Phi\right)^T D_\rho \left(T^\pi(\Phi u) - \Phi w\right) \\ &= \xi \left(\Phi - \gamma P^\pi \Phi\right)^T D_\rho \left(\Phi w - \gamma P^\pi \Phi w - R^\pi\right) + (1-\xi)\Phi^T D_\rho \left(\Phi w - \gamma P^\pi \Phi u - R^\pi\right),\end{aligned}$$

where D_ρ is a diagonal matrix with elements ρ. To find an extrema, we set $\frac{\partial L_{H_2}(w)}{\partial w}$ to 0 and solve for w:

$$\begin{aligned}&\left[\xi \left(\Phi - \gamma P^\pi \Phi\right)^T D_\rho \left(\Phi - \gamma P^\pi \Phi\right) + (1-\xi)\Phi^T D_\rho \Phi\right] w = \\ &\qquad \left[\xi \left(\Phi - \gamma P^\pi \Phi\right)^T D_\rho + (1-\xi)\Phi^T D_\rho\right] R^\pi + (1-\xi)\Phi^T D_\rho (\gamma P^\pi \Phi) u.\end{aligned}$$

By first adding and subtracting $(1-\xi)\Phi^T D_\rho(\gamma P^\pi \Phi)w$ from the left-hand side of the equation and then simplifying both sides of the equation using $\xi\left(\Phi - \gamma P^\pi \Phi\right) + (1-\xi)\Phi = \left(\Phi - \xi\gamma P^\pi \Phi\right)$, we get:

$$\begin{aligned}&\left[\left(\Phi - \xi\gamma P^\pi \Phi\right)^T D_\rho \left(\Phi - \gamma P^\pi \Phi\right) + (1-\xi)\Phi^T D_\rho \left(\gamma P^\pi \Phi\right)\right] w = \\ &\qquad \left(\Phi - \xi\gamma P^\pi \Phi\right)^T D_\rho R^\pi + (1-\xi)\Phi^T D_\rho (\gamma P^\pi \Phi) u.\end{aligned}$$

Finally, we enforce the fixed point constraint $u = w$ to get the final result:

$$\left(\Phi - \xi\gamma P^\pi \Phi\right)^T D_\rho \left(\Phi - \gamma P^\pi \Phi\right) w = \left(\Phi - \xi\gamma P^\pi \Phi\right)^T D_\rho R^\pi.$$

This is a least-squares problem with the form $A_{H_2} w = b_{H_2}$ where:

$$
\begin{aligned}
A_{H_2} &= \Phi^T (I - \xi\gamma P^\pi)^T D_\rho (I - \gamma P^\pi)\Phi \\
&= \xi A_{BR} + (1-\xi) A_{FP} \\
b_{H_2} &= \Phi^T (I - \xi\gamma P^\pi)^T D_\rho R^\pi \\
&= \xi b_{BR} + (1-\xi) b_{FP}.
\end{aligned}
$$

By definition, this technique returns the BR solution when $\xi = 1$ and the FP solution when $\xi = 0$. Importantly, only one $K \times K$ matrix and one length K vector are required. The incremental update given double samples $\langle s, \pi(s), r', s' \rangle$ and $\langle s, \pi(s), r'', s'' \rangle$ has the form:

$$
\begin{aligned}
\hat{A}_{H_2} &= \hat{A}_{H_2} + \rho(s)(\phi(s) - \xi\gamma\phi(s'))(\phi(s) - \gamma\phi(s''))^T \\
\hat{b}_{H_2} &= \hat{b}_{H_2} + \rho(s)(\phi(s) - \xi\gamma\phi(s'))r'.
\end{aligned}
$$

It is worthwhile noting that these updates are nearly identical to those for $\hat{A}_{BR}$ and $\hat{b}_{BR}$ except for the extra parameter ξ.

4.2.4 Difference Between H_1 and H_2 Methods

It is useful to elucidate the differences between the hybrid algorithms. The most immediate difference is that the H_1 method requires three matrices and two vectors whereas the H_2 method only uses one matrix and one vector. We show later in Chapter 6 that when a subset of basis functions from Φ is used (i.e. using $k' \leq K$ columns from Φ), the H_1 method still requires forming $K \times K$ matrices whereas the H_2, BR, and FP algorithms only need $k' \times k'$ storage. This can severely limit the use of algorithm H_1 as it will not scale up to large numbers of basis functions.

To make the comparison of the hybrid methods more obvious, the least-squares equations can be rewritten as follows:

$$A_{FP} = A_{BR} - \gamma^2 \Phi^T (P^\pi)^T D_\rho P^\pi \Phi + \gamma \ \Phi^T (P^\pi)^T D_\rho \Phi$$

$$A_{H_1} = A_{BR} - (1-\xi)\gamma^2 \Phi^T (P^\pi)^T D_\rho P^\pi \Phi + (1-\xi)\gamma^2 \Phi^T (P^\pi)^T D_\rho \Pi_\rho P^\pi \Phi$$

$$A_{H_2} = A_{BR} - (1-\xi)\gamma^2 \Phi^T (P^\pi)^T D_\rho P^\pi \Phi + (1-\xi)\gamma \ \Phi^T (P^\pi)^T D_\rho \Phi.$$

Matrices A_{BR} and A_{H_1} are symmetric by definition whereas A_{FP} and A_{H_2} (except when $\xi = 1$) are not symmetric. Consider the extreme values of ξ. Both A_{H_1} and A_{H_2} are clearly the same as A_{BR} when $\xi = 1$. When $\xi = 0$, it is obvious that A_{H_2} and A_{FP} are identical. It is less obvious that algorithm H_1 produces the same solution w to the least-squares minimization as algorithms H_2 and FP when $\xi = 0$, but this can in fact be shown. The interesting case occurs when $0 < \xi < 1$ because the hybrid solutions differ. Notice the only difference between A_{H_1} and A_{H_2} is in the final term shown above. The final term in A_{H_2} is $\gamma \Phi^T (P^\pi)^T D_\rho \Phi$, while A_{H_1} includes the same term times its transpose. This occurs during the least-squares derivation of A_{H_1}.

As shown in Equation 4.3, the H_1 method can be written as a minimization over the sum of two loss functions: the norm of the Bellman residual and the norm of the projected Bellman residual. Algorithm H_2 uses a slightly different way of enforcing the fixed point loss function. The fixed point constraint is applied only after the optimization problem is solved.

4.2.5 Other Possible Algorithms

The two proposed hybrid algorithms *implicitly* constrain the Bellman residual by the choice of the parameter ξ. This constraint could be made explicit. The problem would be to minimize the projection of the Bellman residual subject to an inequality constraint on the Bellman residual (either on its magnitude or component-wise). This constrained optimization would take the form:

$$\min_{w} \quad \|A_{FP}w - b_{FP}\|_{\rho}$$
$$\text{subject to: } \|A_{BR}w - b_{BR}\|_{\rho} \leq \delta$$
$$\text{or:} \quad \pm (A_{BR}w - b_{BR}) \leq \Delta.$$

The parameters $\delta \in \mathbb{R}^{+}$ and $\Delta \in \mathbb{R}^{K+}$ must be set appropriately based on the minimal value of the Bellman residual magnitude attained using the BR method. We point out the possibility of explicitly controlling the Bellman residual to be thorough. However, since this increases the computational complexity, we limit our discussion to the two simple least-squares algorithms H_1 and H_2.

4.3 Analysis

4.3.1 Projection of the Target Function

The first three approximate policy evaluation techniques were shown to be images of the target function V^{π} under different projection operations [88]. More specifically, each method $X = \{OPT, BR, FP\}$ produces an approximate value function with the following form: $\hat{V} = \Phi w = \Phi A_X^{-1} b_X = \Phi(\Phi^T D_X \Phi)^{-1} \Phi^T D_X V^{\pi}$. The matrix D_X controls the weighting of the projection and takes on the following values [88]:

$$D_{OPT} = D_{\rho}$$
$$D_{BR} = (I - \gamma P^{\pi})^T D_{\rho} (I - \gamma P^{\pi})$$
$$D_{FP} = D_{\rho}(I - \gamma P^{\pi}).$$

The hybrid methods have a similar characterization:

$$D_{H_1} = (I - \gamma P^{\pi})^T D_{\rho} (\xi I + (1 - \xi)\Pi_{\rho})(I - \gamma P^{\pi})$$
$$D_{H_2} = (I - \xi\gamma P^{\pi})^T D_{\rho}(I - \gamma P^{\pi}).$$

4.3.2 Geometry of the Bellman Equation

Each approximate policy evaluation algorithm uses the Bellman equation in different ways to compute a value function. There is an intuitive geometric perspective to the algorithms when using linear function approximation. We expand on the original presentation of this perspective [56].

The Bellman equation with linear function approximation has three components: $\hat{V}$, $T^{\pi}(\hat{V})$, and $\Pi_{\rho}T^{\pi}(\hat{V})$. These components geometrically form a triangle where $\hat{V}$ and $\Pi_{\rho}T^{\pi}(\hat{V})$ reside in the space spanned by Φ while $T^{\pi}(\hat{V})$ is, in general, outside this space. This is illustrated in the leftmost triangle of Figure 4.3. The three-dimensional space in the figure is the space of exact value functions while the two-dimensional plane represents the space of approximate value functions in $[\Phi]$. The angle between subspace $[\Phi]$ and the vector $T^{\pi}(\hat{V}) - \hat{V}$ is denoted θ. The BR and FP solutions minimize the length of different sides of the triangle. The second triangle in Figure 4.3 shows the BR solution, which minimizes the length of $T^{\pi}(\hat{V}) - \hat{V}$. The third (degenerative) triangle shows the FP solution, which minimizes the length of $\Pi_{\rho}T^{\pi}(\hat{V}) - \hat{V}$. This length is 0 which means $\theta_{FP} = 90°$. The fourth triangle shows the H solution, which minimizes a combination of the lengths of the two sides. In general, θ_{H} lies between θ_{BR} and $90°$. The hybrid solution allows for controlling the shape of this triangle. We purposefully drew the triangles in Figure 4.3 suggestively to not only accentuate their angles, but also to emphasize that the length of the Bellman residual ($T^{\pi}(\hat{V}) - \hat{V}$) can become large at times for the FP method. By including the norm of the Bellman residual in their objective functions, hybrid algorithms can protect against such large residual vectors. They also have the flexibility of finding solutions that are almost fixed points but have more desirable properties (smaller Bellman residuals).

4.3.3 "Backward Bootstrapping"

In RL, bootstrapping refers to the process of updating estimated values based on subsequent estimated values. (i.e. making the value of a state look like its successor states). It

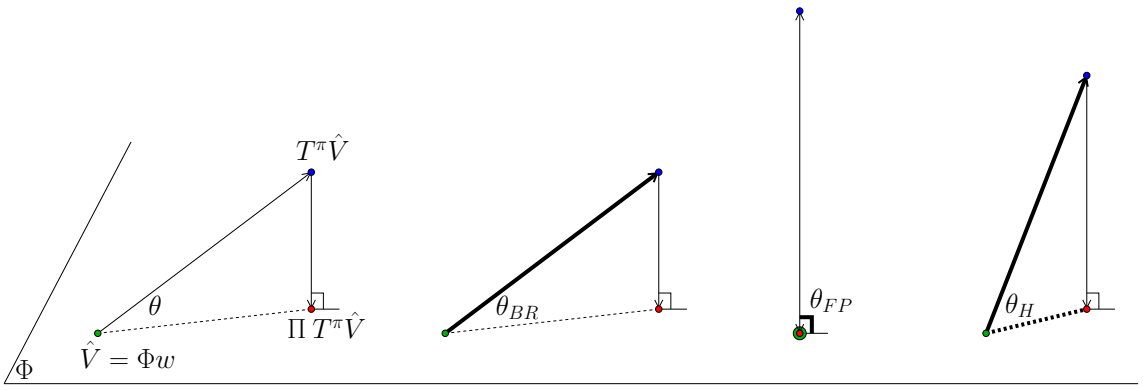


Figure 4.3. The triangle on the left shows the general form of the Bellman equation. The other three triangles correspond to the different approximate policy evaluation algorithms where the bold lines indicate what is being optimized.

has been pointed out by Dayan [26] and again recently by Sutton et al. [99] that algorithms using the $L_{BR}(w)$ loss function perform "backward bootstrapping," i.e., they also make a state look like its preceding states. To visualize this effect, we consider the simple example shown in Figure 4.4. This is the sample problem used by Dayan [26] and Sutton et al. [99]. There are four states, an equal likelihood of starting in state A1 or A2, and deterministic transitions to terminal states ending with a reward of 1 or 0. There are three basis functions: one distinguishing state B, one distinguishing state C, and one representing both states A1 and A2. Thus, states A1 and A2 are indistinguishable based on the features.

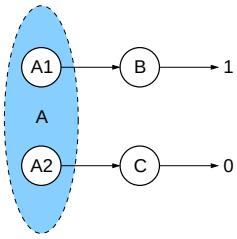


Figure 4.4. Small example from [26, 99] to illustrate backward bootstrapping. States A1 and A2 have the same feature representation.

Since state A1 (which should have value 1) and state A2 (which should have value 0) are indistinguishable given the features and since they occur with equal likelihood, they are assigned a value of $\frac{1}{2}$ by the FP, BR, and hybrid algorithms. The least-squares algorithms

State	V^π	$\hat{V}_{FP}$	$\hat{V}_{H_2}(\xi = \frac{1}{3})$	$\hat{V}_{H_2}(\xi = \frac{2}{3})$	$\hat{V}_{BR}$
A1	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
A2	0	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
B	1	1	$\frac{7}{8}$	$\frac{4}{5}$	$\frac{3}{4}$
C	0	0	$\frac{1}{8}$	$\frac{1}{5}$	$\frac{1}{4}$

Table 4.2. Value functions associated with the example in Fig. 4.4.

differ though in the values they assign to states B and C. The FP algorithm avoids backward bootstrapping and assigns the appropriate values of 1 to state B and 0 to state C. The BR algorithm assigns values of $\frac{3}{4}$ and $\frac{1}{4}$. This is because having a Bellman residual of $\pm\frac{1}{4}$ at all four states is cheaper according to loss function $L_{BR}(w)$ than having a Bellman residual of $\pm\frac{1}{2}$ at just the two states A1 and A2. The hybrid algorithms find solutions between the FP and BR extremes. The exact value function and approximate value functions using the various least-squares methods are shown in Table 4.2.

Sutton et al. [99] argue that backward bootstrapping is to be avoided. We are not aware of any theoretical argument for avoiding Bellman residual algorithms. As mentioned in Section 4.2.1, the argument for using the Bellman residual loss function has to do with BR's stability [74] and that associated performance bounds are described in terms of the Bellman residual [109] (smaller Bellman residuals result in tightening of the bounds).

4.4 Laplacian Regularization

This chapter describes four least-squares algorithms for approximate policy evaluation. The algorithms compute approximate value functions that minimize a loss function associated with the Bellman residual. Here, we include an additional loss function that *regularizes* the solution. Regularization entails using an additional loss function to either help solve an ill-posed problem or prevent overfitting. The concept of well-posedness of a problem dates back to Hadamard [40]. He deemed problems well-posed if (1) a solution exists, (2) the solution is unique, and (3) the solution depends continuously on the data. If a problem

is not well-posed, then it is ill-posed. There is also a similar notion of a problem being ill-conditioned. For example, in least-squares problems $Aw = b$, if the solution w changes drastically for minor changes to the matrix A, then the problem is ill-conditioned. Regularization helps to find a meaningful solution in the face of such challenges. One of the most popular techniques for regularizing ill-posed problems is Tikhonov regularization [104]. In addition to the standard least-squares minimization $\min_w \|Aw - b\|^2$, an additional term is included:

$$\min_w \left(\|Aw - b\|^2 + \beta_r \|\Gamma w\|^2 \right),$$

where $\beta_r \geq 0$ is the regularization parameter managing the trade-off between loss functions and Γ is an appropriately chosen Tikhonov matrix. Often times the identity matrix is used ($\Gamma = I$) which gives preference to coefficients w with a small L_2 norm. The solution to the regularized minimization problem is $w^* = \left(A^T A + \beta_r \Gamma^T \Gamma\right)^{-1} A^T b$.

Here we consider Laplacian-based regularization. It is a *data-dependent* form of regularization that uses the graph Laplacian [7]. This is the same graph Laplacian used to produce PVFs and diffusion wavelets. Laplacian-based regularization has been applied with great success to semi-supervised learning problems where the geometric structure of unlabeled data points can be exploited [7]. To understand how the graph Laplacian provides regularization, consider again the Dirichlet sum which was described in Equation 3.1 of Section 3.2. Given function f, the Dirichlet sum is $\langle f, Lf \rangle = \sum_{u \sim v} W(u, v) \left(f(u) - f(v)\right)^2$. The Dirichlet sum is large when f is not smooth according to the structure of the graph. For functions that are smooth, the Dirichlet sum is small. Thus, the Laplacian can be used to penalize (regularize) functions that are not smooth according to the structure of the MDP state space encoded in the graph.

We described Tikhonov regularization above using the matrix Γ and the loss function $\|\Gamma w\|^2$. Laplacian-based regularization can be described in this manner using $\Gamma = L\Phi$. As a concrete example, consider the H_2 least-squares algorithm. H_2's loss function $L_{H_2}(w)$ is

based on the Bellman residual and the projected Bellman residual. We augment that loss function with a Laplacian-based regularization (LR) term as follows:

$$w_{H_2,LR} = \underset{w' \in \mathbb{R}^K}{\operatorname{argmin}} \left(\frac{\xi}{2} \|T^\pi(\Phi w') - \Phi w'\|_\rho^2 + \frac{1-\xi}{2} \|T^\pi(\Phi u) - \Phi w'\|_\rho^2 + \frac{\beta_r}{2} \|L\Phi w'\|_\rho^2 \right), \tag{4.4}$$

where $\beta_r \in \mathbb{R}^+$ is a parameter controlling the influence of the regularization term. The loss function $\|L\Phi w'\|_\rho^2$ penalizes non-smooth value functions $\Phi w'$. Following the same steps shown in Section 4.2.3, one can show $w_{H_2,LR} = A_{H_2,LR}^{-1} b_{H_2,LR}$ where:

$$\begin{aligned} A_{H_2,LR} &= (\Phi - \xi\gamma P^\pi \Phi)^T D_\rho (\Phi - \gamma P^\pi \Phi) + \beta_r \Phi^T L D_\rho L \Phi \\ b_{H_2,LR} &= (\Phi - \xi\gamma P^\pi \Phi)^T D_\rho R^\pi. \end{aligned} \tag{4.5}$$

Notice that $b_{H_2,LR} = b_{H_2}$ and $A_{H_2,LR} = A_{H_2} + \beta_r \Phi^T L D_\rho L \Phi$. Given a sample $\langle s, \pi(s), r, s' \rangle$ from the MDP, estimates of the matrix $A_{H_2,LR}$ and vector $b_{H_2,LR}$ can be formed using the following updates:

$$\begin{aligned} \hat{A}_{H_2,LR} &\leftarrow \hat{A}_{H_2,LR} + \rho(s) \left[(\phi(s) - \xi\gamma\phi(s'))(\phi(s) - \gamma\phi(s'))^T + \beta_r g(s) g(s)^T \right] \\ \hat{b}_{H_2,LR} &\leftarrow \hat{b}_{H_2,LR} + \rho(s)(\phi(s) - \xi\gamma\phi(s'))r. \end{aligned}$$

The term $g(s)$ in the updates is computed as:

$$\begin{aligned} g(s) &\leftarrow L(s,s)\phi(s) \\ g(s) &\leftarrow g(s) + L(s, s_{nbr})\phi(s_{nbr}) \quad \forall \{ s_{nbr} | s_{nbr} \neq s \ \wedge \ s \sim s_{nbr} \text{ in graph} \}. \end{aligned}$$

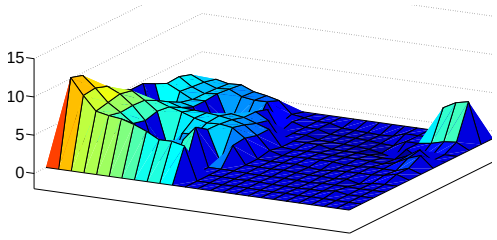
A common assumption is that MDP state space graphs are sparsely connected. This means that any state s has at most a few neighboring states s_{nbr} in the graph. In this case, the time to compute $g(s)$ is negligible. Of course, if the basis functions $\phi(s)$ are the PVFs,

then the eigendecomposition $L\Phi = \Phi\Lambda$ can be exploited to simplify the computation as $g(s) \leftarrow \Lambda\phi(s)$.

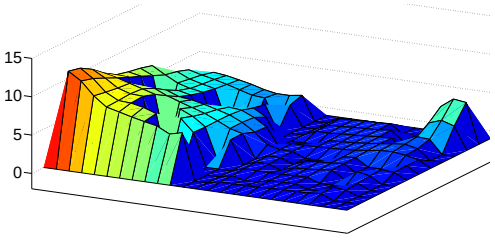
When the value function V^π can be computed exactly, there is no need for regularization. But when the value function is approximated using function approximation and a finite set of samples from a stochastic MDP, regularization can have a positive effect on the solution. To demonstrate this, consider a two-room grid MDP with 243 states and a reward of $+1$ in opposite corners of the two rooms. We set the probability of an action succeeding to 0.6; failed actions move the agent in one of the other 3 directions. The discount factor was set to $\gamma = 0.95$. The optimal value function for this problem is shown in Figure 4.5(e). We generated 500 samples (100 episodes starting in a random start state and lasting 5 steps each) using an optimal policy. Not every state in the MDP was visited during this exploration. The location of the sampled states is shown in Figure 4.5(f). We used the smoothest 50 proto-value functions for basis functions and the FP least-squares algorithm with Laplacian regularization. The top four plots in Figure 4.5 show the effect regularization has on the approximate value functions. It can be seen that increasing the regularization parameter β_r smooths out the approximate value function but at too large a value "over-regularizes" the solution. An automated approach for setting β_r is to use so-called L-curves. The L-curve is a plot of the original loss function (e.g. $L_{H_2}(w)$) on one axis and the penalty function (e.g. $\|L\Phi w\|_\rho^2$) on the other axis for several values of the parameter β_r. The original loss function dominates when β_r is small and the penalty function dominates when β_r is large (thus the shape of the curve is a L). It is common to select a value for β_r somewhere near the inflection point of the L-curve.

4.5 Algorithmic Details

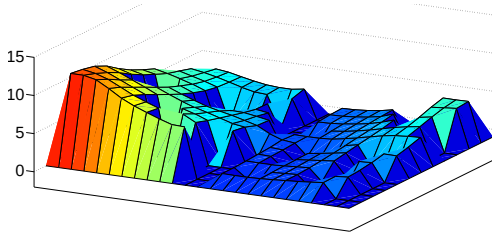
The previous sections of this chapter described the objective functions for regularized least-squares RL algorithms. Here we provide pseudocode for approximate policy evalu-



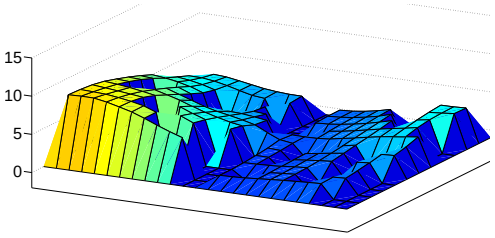
(a) $\hat{V}_{FP}, \beta_r = 0$



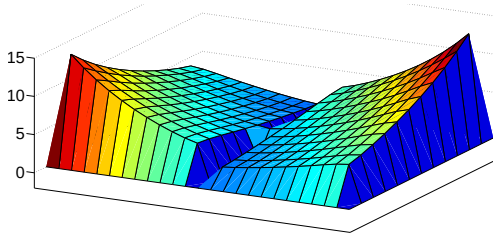
(b) $\hat{V}_{FP}, \beta_r = 0.1$



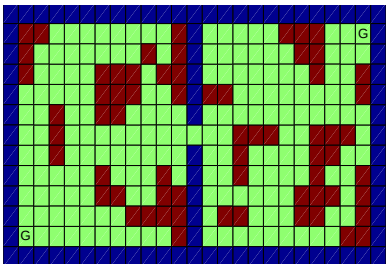
(c) $\hat{V}_{FP}, \beta_r = 1.0$



(d) $\hat{V}_{FP}, \beta_r = 10.0$



(e) V^*



(f) Location of samples (green is observed, red is unobserved)

Figure 4.5. The effect of Laplacian regularization on a two-room MDP with a set of 500 samples.

ation and approximate policy iteration. In both cases, we assume the value function (or action-value function) is approximated using a batch of samples from a MDP.

4.5.1 Approximate Policy Evaluation

Given a batch of samples of the form $\{s, \pi(s), r, s'\}$, least-squares methods for approximate policy evaluation use one of the objective functions (BR, FP, H_1, or H_2) to generate an approximation of V^π. We describe pseudocode here using the hybrid H_2 objective function because it naturally encompasses the FP ($\xi = 0$) and BR ($\xi = 1$) methods. It is obvious how the pseudocode can change to accommodate the H_1 objective function.

Assuming a K dimensional feature vector ϕ, the algorithm builds up sample-based statistics for the matrix $\hat{A} \in \mathbb{R}^{K \times K}$ and vector $\hat{b} \in \mathbb{R}^K$. The approximate value function is $\hat{V} = \Phi \hat{A}^{-1} \hat{b}$. Pseudocode for the H_2 method is shown in Algorithm 2.

Algorithm 2: `Hybrid Least-Squares Policy Evaluation Method (H2)`

Input: $\{s_i, r_i, s'_i\}_{i=1}^n$, n samples generated using policy π
$\phi : S \rightarrow \mathbb{R}^K$, basis function
$\rho : S \rightarrow \mathbb{R}^+$, weighting over the states
$\gamma \in [0, 1]$, discount factor
$\xi \in [0, 1]$, hybrid parameter ($\xi = 0$ is FP, $\xi = 1$ is BR)
L, graph Laplacian defined over states $\{s_i\}_{i=1}^n$ (graph edges denoted with $\sim$)
$\beta_r \in \mathbb{R}^+$, Laplacian-based regularization parameter

Output: w, weight vector such that $\hat{V}(s) = \phi(s)^T w$

Initialize $\hat{A} \leftarrow \mathbf{0}$, $\hat{b} \leftarrow \mathbf{0}$

for $i = 1$ to n **do**
$\hat{A} \leftarrow \hat{A} + \rho(s_i) \left[(\phi(s_i) - \xi\gamma\phi(s'_i)) \; (\phi(s_i) - \gamma\phi(s'_i))^T + \beta_r \, g(s_i) g(s_i)^T \right]$
$\hat{b} \leftarrow \hat{b} + \rho(s_i) \, (\phi(s_i) - \xi\gamma\phi(s'_i)) \, r_i$

where: $g(s_i) \leftarrow L(s_i, s_i) \, \phi(s_i)$
$g(s_i) \leftarrow g(s_i) + L(s_i, s_{nbr}) \, \phi(s_{nbr}) \qquad \forall \{ s_{nbr} | s_{nbr} \neq s_i \wedge s_i \sim s_{nbr} \}$

end for
$w \leftarrow \hat{A}^{-1} \hat{b}$

4.5.2 Approximate Policy Iteration

The least-squares policy iteration (LSPI) algorithm is a data efficient control learning algorithm proposed by Lagoudakis and Parr [56]. LSPI is an iterative algorithm. At each

iteration, a policy is evaluated resulting in an approximate action-value function $\hat{Q} = \Phi w$. The greedy policy associated with $\hat{Q}$ is then used in the next iteration. The algorithm terminates when the approximation $\hat{Q}$ converges or when a specified maximum number of iterations has been reached.

The algorithm uses a batch of MDP samples of the form $\{s_i, a_i, r_i, s'_i\}_{i=1}^{n}$. The distribution of the samples can have a significant impact on the least-squares solution. To account for this distribution, one can weight each sample separately using $\rho(s_i, a_i)$. The pseudocode for LSPI with Laplacian-based regularization and the H_2 objective function is shown in Algorithm 3.

Algorithm 3: Hybrid Least-Squares Policy Iteration Method (H_2)

Input: $\{s_i, a_i, r_i, s'_i\}_{i=1}^{n}$, MDP samples
$\phi : S \times A \rightarrow \mathbb{R}^K$, basis function
$\rho : S \times A \rightarrow \mathbb{R}^+$, weighting over state-action pairs (can change each iteration)
$\gamma \in [0, 1]$, discount factor
$\xi \in [0, 1]$, hybrid parameter ($\xi = 0$ is FP, $\xi = 1$ is BR)
$w_0 \in \mathbb{R}^K$, (optional) initial weight vector
L_a, $|A|$ graph Laplacians, each defined over states occurring with action a
$\beta_r \in \mathbb{R}^+$, Laplacian-based regularization parameter
Output: w, weight vector such that $\hat{Q}(s, a) = \phi(s, a)^T w$

$w \leftarrow w_0$ (or initialized randomly if w_0 is not given)

while (not converged) **do**
  Initialize $\hat{A} \leftarrow \mathbf{0}$, $\hat{b} \leftarrow \mathbf{0}$
  for $i = 1$ to n **do**
    $a^* \leftarrow \text{argmax}_{a \in A_{s'_i}} \left(\phi(s'_i, a)^T w\right)$
    $\hat{A} \leftarrow \hat{A} + \rho(s_i, a_i)\ (\phi(s_i, a_i) - \xi\, \gamma\, \phi(s'_i, a^*))\ (\phi(s_i, a_i) - \gamma\, \phi(s'_i, a^*))^T + \ldots$
      $\rho(s_i, a_i)\, \beta_r\, g(s_i, a_i) g(s_i, a_i)^T$
    $\hat{b} \leftarrow \hat{b} + \rho(s_i, a_i)\ (\phi(s_i, a_i) - \xi\, \gamma\, \phi(s'_i, a^*))\ r_i$
    where: $g(s_i, a_i) \leftarrow L_{a_i}(s_i, s_i)\, \phi(s_i, a_i)$
      $g(s_i, a_i) \leftarrow g(s_i, a_i) + L_{a_i}(s_i, s_{nbr})\, \phi(s_{nbr}, a_i), \quad \forall \{s_{nbr} | s_{nbr} \neq s_i \wedge s_i \sim s_{nbr}\}$
  end for
  $w \leftarrow \hat{A}^{-1} \hat{b}$
end while

Mahadevan proposed an overall framework for combining (1) sample generation, (2) representation learning using graph-based basis functions, and (3) control learning. This links together all aspects of the RL problem. The framework is called Representation

Sample Collection Phase

1. Generate a data set $\mathcal{D}$ of "state-action-reward-nextstate" transitions (s_t, a_t, r_t, s_{t+1}) using some policy (e.g. a random walk).
2. Sparsification Step: Subsample a set of transitions $\mathcal{D}_s$ from $\mathcal{D}$ by some method (e.g. randomly or greedily).

Representation Learning Phase

3. Construct a *diffusion model* from $\mathcal{D}_s$ consisting of an undirected graph $G = (V, E, W)$ with edge set E and weight matrix W. Each vertex $v \in V$ corresponds to a visited state. Given an appropriate local distance metric $d(\cdot, \cdot)$, edges are inserted between a pair of vertices x_i and x_j using either a k nearest neighbor or an ϵ nearest neighbor algorithm. Edge weights are assigned $W(i, j) = \alpha(i) \exp\left(-\frac{d(x_i, x_j)}{\sigma}\right)$ where $\sigma > 0$ is a parameter and α is a specified weight function.
4. Form either:
   (a) the graph Laplacian $L = D - W$ where D is a diagonal matrix of the row sums of W.
   (b) the diffusion operator $T = D^{-0.5} W D^{-0.5}$.
5. Compute the K "smoothest" eigenvalues (λ_i) and eigenvectors (ϕ_i) of L or compute the diffusion wavelet tree from T and select the K most global scaling and wavelet functions. The basis function matrix is $\Phi = [\phi_1, \phi_2, \ldots, \phi_K]$.

Control Learning Phase

6. Use a parameter estimation method such as hybrid least-squares policy iteration or Q-learning [98] to find a (good) policy represented by the action-value function $\hat{Q} = \Phi w$.

Figure 4.6. The RPI framework for learning representation and control in MDPs.

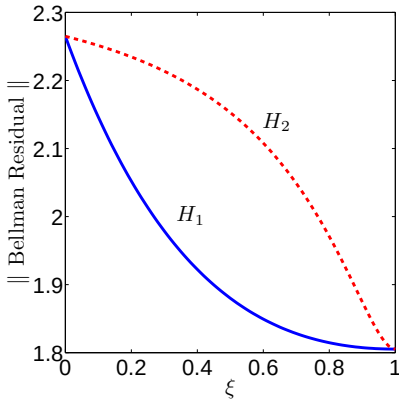
Policy Iteration (RPI) [64] and is described in Figure 4.6. The experiments demonstrating RPI [67] used a random policy for the sample generation component, proto-value functions for the representation learning component, and LSPI for the control learning component.

4.6 Experiments

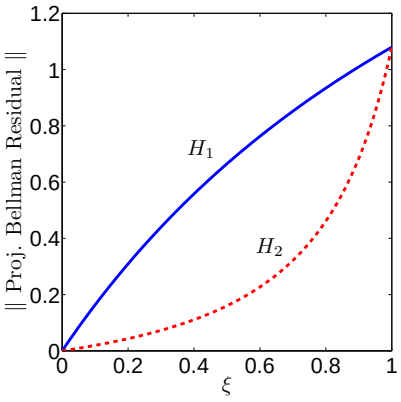
Reconsider the chain MDP from Figure 4.1 and the basis functions from Figure 4.2. The optimal policy for this MDP with a discount factor $\gamma = 0.99$ is $\pi^* = \textit{RRRLLL}$. Starting from initial policy $\pi_0 = \textit{LLLLLL}$, the BR method results in an approximate value function $\hat{V}_{BR}^{\pi_0}$ whose greedy policy is also $\pi_1 = \textit{LLLLLL}$. In other words, after one round of policy iteration, it converges on the initial policy. The FP algorithm produces an approximate value function $\hat{V}_{FP}^{\pi_0}$ whose greedy policy is $\pi_1 = \textit{RRRRRR}$. Hybrid algorithms find solutions between these two extremes. We ranged the value of ξ from 0 to 1 and computed $\hat{V}_{H_1}^{\pi_0}$ and $\hat{V}_{H_2}^{\pi_0}$ using the equations in Section 4.2 (the transition matrix and reward function were not sampled but rather were used explicitly). We also recorded the norm of the Bellman residual, the norm of the projected Bellman residual, the angle between the Bellman residual and the space spanned by the basis functions Φ, and the greedy policies associated with the approximate value functions. The results are shown in Figure 4.7 using a uniform distribution ρ; however, the results are very similar when setting ρ to be the invariant distribution of P^{π_0}. Note the trade-off between the Bellman residual and the projected Bellman residual for different values of ξ in Figures 4.7(a) and 4.7(b). In Figure 4.7(b), the curve associated with method H_2 is beneath that of method H_1. This indicates algorithm H_2 places more weight on minimizing the projected Bellman residual compared to algorithm H_1. Also, note that the greedy policies in Figures 4.7(d) and 4.7(e) run the full gamut from $\textit{RRRRRR}$ at $\xi = 0$ to $\textit{LLLLLL}$ at $\xi = 1$.

We compared all methods on a 10×10 grid MDP. The MDP has 100 states, 4 actions that have probability 0.9 of success (an unsuccessful action resulted in a transition in one of the other three directions), a 0.95 discount factor, and a reward of +1 in one corner and +2 in the diagonal corner. Fifteen Laplacian eigenvectors [64] were used as basis functions.

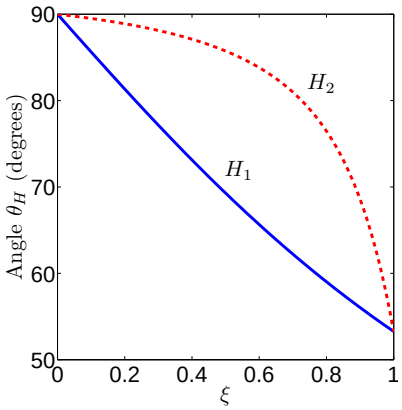
We ran 500 trials. Each trial began with a randomly initialized policy, then policy iteration was run using each policy evaluation method until the weight vector converged or 500 iterations were reached. The model was used during policy iteration to avoid any



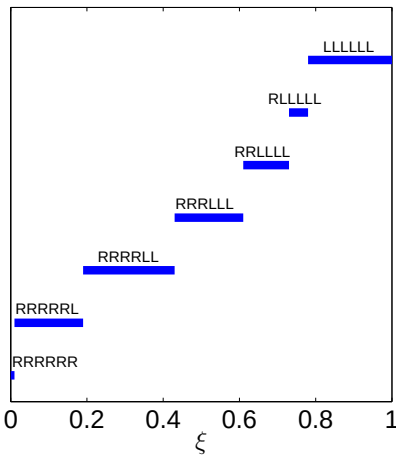
(a) Bellman residual vs. ξ.



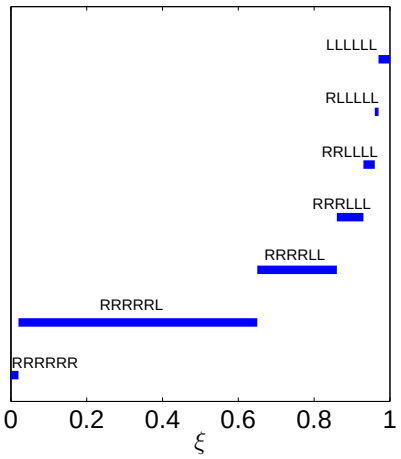
(b) Proj. Bellman residual vs. ξ.



(c) Angle θ_H vs. ξ.



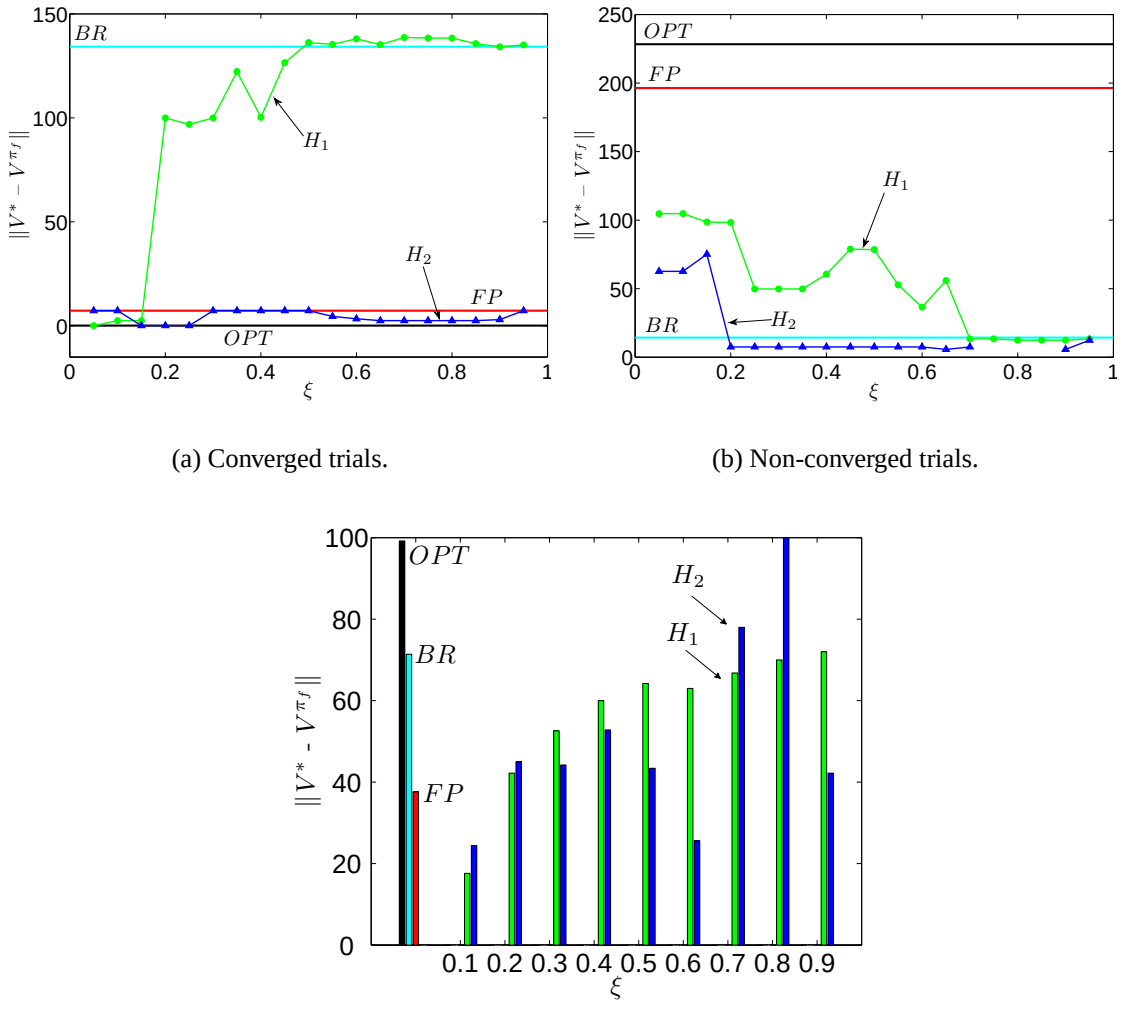
(d) Greedy policy w.r.t. $\hat{V}^{\pi_0}_{H_1}$.



(e) Greedy policy w.r.t. $\hat{V}^{\pi_0}_{H_2}$.

Figure 4.7. Results of approximate policy evaluation using the hybrid least-squares algorithms for the MDP in Figure 4.1.

difficulty comparing the various methods due to sampling. The result of policy iteration is a final policy π_f. We evaluate these policies by computing V^{π_f} exactly and comparing it with V^*. The results, which are broken into the trials that converged and those that did not converge, are shown in Figure 4.8.



(a) Converged trials.

(b) Non-converged trials.

(c) % of trails that converged.

Figure 4.8. Results of 500 policy iteration trials for the grid MDP. The results are divided into those trials that converged (a) versus those that did not converge (b). The median value of $\|V^* - V^{\pi_f}\|$ is plotted versus ξ, where π_f is the final policy attained when policy iteration terminates. The percentage of trials that converged is shown in (c).

The BR algorithm converged almost twice as often as the FP algorithm (71.4% compared to 37.6%). However, when the BR method converged, it happened after only 8.5 rounds of policy iteration on average. That strongly contrasts with the fixed point method's average of 89.1 rounds of policy iteration until convergence. Since the BR method tends to make small changes to the value function between rounds of policy iteration, it is not surprising that that this early convergence (starting from a random policy) leads to very suboptimal policies. It is interesting that the BR method discovered better policies when policy iteration did *not* converge. On the other hand, when the FP method converged it found excellent policies (small values of $\|V^* - V^{\pi_f}\|$).

The policies found by algorithm H_1 had a general linear trend between $\xi = 0$ (FP) and $\xi = 1$ (BR). The policy iteration convergence rate had a similar effect. The convergence rate was not nearly as predictable for algorithm H_2. In fact, at $\xi = 0.8$, all 500 trials converged. The most interesting aspect of this experiment is the excellent performance of algorithm H_2. The method produced good policies regardless of convergence and across all ξ values.

We have presented hybrid least-squares algorithms for approximating value functions, but the same idea holds for approximating action-value functions. We tested all policy evaluation methods on the problem of learning an approximate action-value function for Tetris. Ten basis functions over state-action pairs (s, a) were used. The first four are for the current state s: maximum height, number of holes, sum of absolute height differences between adjacent columns, and the mean height. The next four basis functions are the change in the value of the first four features after taking action a from s. The last two are the change in the score and a constant 1. This feature set was proposed by Lagoudakis et al. [57].

Forty episodes of data ($\sim$30,000 samples) were generated using an initial policy greedy with respect to weight vector w^{π_0} = [-1, -10, -1, -1, -2, -11, -2, -2, 50, 10]^T. We ran policy iteration starting from w^{π_0} until the weight vector converged or 100 iterations were

reached. Instead of generating double samples to form unbiased estimates of A and b, we used the model to compute the expectation over next-states and actions. For Tetris, each action results in seven equally likely next-states corresponding to the seven Tetris pieces. This method of using the model for transitions instead of a sample from the model was described by Lagoudakis and Parr as **LSTD*Q*-Model** [56].

We tested the learned policies 50 times. Each time, we generated a random ordered set of pieces that *all* policies were forced to place to make the comparison more accurate. This is necessary because Tetris performance can be very sensitive to the exact order of pieces. The average score over the 50 trials is shown in Table 4.3. The initial policy w^{π_0} scored 310 on average. Policy iteration converged in less than 7 iterations for the FP and H_2 methods, whereas the BR and H_1 methods did not converge. The performance split along this division. The final policy computed using the BR method rarely removed a line. This was also the case for policies learned using algorithm H_1 except when $\xi = 0.4$. On the other hand, the policies learned using the FP and H_2 methods performed at least as well as the initial policy and in some cases significantly better. The best policy was computed using algorithm H_2 with $\xi = 0.1$.

Table 4.3. Results of policy iteration for Tetris. An asterisk indicates policy iteration converged.

Technique	Score	Technique	Score
BR	0	FP*	630
H_1, ξ=0.1	15	H_2, ξ=0.1*	800
H_1, ξ=0.2	0	H_2, ξ=0.2*	580
H_1, ξ=0.3	80	H_2, ξ=0.3*	645
H_1, ξ=0.4	295	H_2, ξ=0.4*	515
H_1, ξ=0.5	60	H_2, ξ=0.5*	455
H_1, ξ=0.6	5	H_2, ξ=0.6*	395
H_1, ξ=0.7	5	H_2, ξ=0.7*	370
H_1, ξ=0.8	0	H_2, ξ=0.8*	405
H_1, ξ=0.9	0	H_2, ξ=0.9*	330

4.7 Conclusions

The fixed point (FP) and Bellman residual (BR) algorithms can be combined to form a hybrid approximate policy evaluation algorithm. We proposed two ways to implement hybrid algorithms using least-squares methods, thus improving efficiency over the original incremental algorithm [3]. The two implementations differ in how they handle the fixed point constraint. The first implementation (H_1) enforces the fixed point constraint and then derives a least-squares formula whereas the second implementation (H_2) performs those two steps in reverse. We analyzed the algorithms in terms of projections of the target function and showed that hybrid algorithms have an intuitive geometric interpretation.

Hybrid least-squares algorithms attempt to combine the stability of the BR solution with the improved performance of the FP solution. We presented an example on a chain MDP demonstrating this effect. Policy iteration experiments were conducted on a simple grid MDP so that the quality of the learned policies could be determined analytically. Experiments were also run on the challenging task of learning to play Tetris where learned policies were evaluated empirically. In both domains, the hybrid algorithm H_2 discovered policies that performed much better than the BR and H_1 methods and as well as, and in some instances better than, the FP method. The hybrid algorithm H_2 has the same data structures and computational complexity as the BR and FP methods whereas the H_1 algorithm is more complex. A surprising finding was the H_2 method's robustness for a wide range of ξ values. One would expect that for ξ values close to 1, the difference between the BR and H_2 methods would be minimal. Providing a mechanism for automatically setting ξ is an interesting area for future work.

In Section 4.4, we showed how the least-squares algorithm can be augmented to include Laplacian-based regularization [7]. Laplacian-based regularization penalizes functions that are not smooth according to the structure of the graph. This type of regularization can be useful when the domain is stochastic and relatively few samples are available for learning a policy. Since, at this point, we have used the graph Laplacian for both basis construction

and for regularization, it is worthwhile stepping back and understanding the distinction. The initial work by Mahadevan and Maggioni [67, 63] used the "smoothest" Laplacian eigenvectors and diffusion wavelets as basis functions. This set of basis functions *implicitly* constrains (regularizes) the space of approximate value functions that can be represented. In this dissertation, we think of the Laplacian eigenvectors and diffusion wavelets as *dictionaries* of basis functions. We are free to use any elements, not just the smoothest, from the dictionary to approximate a particular value function (note this perspective is implemented with the basis selection algorithms in Chapter 6). Since any elements from the dictionary can be used, this expands the space of approximate value functions that can be represented. To ensure these functions retain some degree of smoothness, we use the graph Laplacian matrix to *explicitly* regularize the approximate value functions. Thus, our approach offers greater flexibility in the type of value functions that can be learned, while retaining the ability to regularize the solutions.

CHAPTER 5

EFFICIENT BASIS CONSTRUCTION FOR LARGE GRAPHS

A significant challenge for any value function approximation architecture is scalability. For example, a naïve implementation of tile codings becomes infeasible with increasing dimensionality of the state space. This is commonly referred to as the "curse of dimensionality." A natural question to ask, therefore, is how basis functions derived from an operator on a graph scale to large problems. This question can be decomposed into two components: (1) the construction of the graph from samples from a Markov decision process and (2) the generation of the basis functions from the graph.

A graph is constructed from an agent's actual experience in a Markov decision process. Therefore, the size of the graph grows over time as the agent explores the state space. This means that, for discrete domains, the size of the graph is at most the number of unique states visited. Of course, the graph's size could be less if some combination of sampling and/or abstraction is performed. In domains with continuous state variables, the graph-based framework has the nice property that the graph reflects the *intrinsic* dimensionality of the problem. While a problem may nominally be high dimensional, there is often lower dimensional structure explaining the data. For example, the graph framework was originally applied to the more general problem of nonlinear dimensionality reduction. In Tenenbaum et al.'s work [101], the vertices of the graph were high dimensional images (faces) but in fact could be explained by just three dimensions (up-down pose, left-right pose, and lighting). This same type of structure exists in many interesting control problems such as high degree of freedom robotic systems that are constrained by their dynamics.

Therefore, the actual construction of a graph from experience in a Markov decision process in principle seems feasible for many problems.

The more significant problem is the computational complexity of generating basis functions from a graph with $|V|$ vertices. It is reasonable to assume the graph is sparse, meaning there are $O(|V|)$ entries as opposed to a dense graph with $O(|V|^2)$ entries. A dense graph corresponds to an environment where an agent could transition from any state to any other state in a small number of steps. Domains with such little structure are uninteresting. Therefore, in the context of Laplacian eigenfunctions, the computational complexity depends on the amount of time it takes to generate K eigenvectors of a sparse, Hermitian matrix. We assume K is much smaller than $|V|$. This is an extremely important problem in linear algebra with many applications in science and engineering. As such, there has been extensive research in finding faster algorithms. The current method used in Matlab^{TM} is an implicitly restarted Arnoldi method that can compute a few eigenvectors for sparse matrices with $O(10^5)$ rows. Parallel algorithms are a current research trend given the growing number of multi-processor machines.

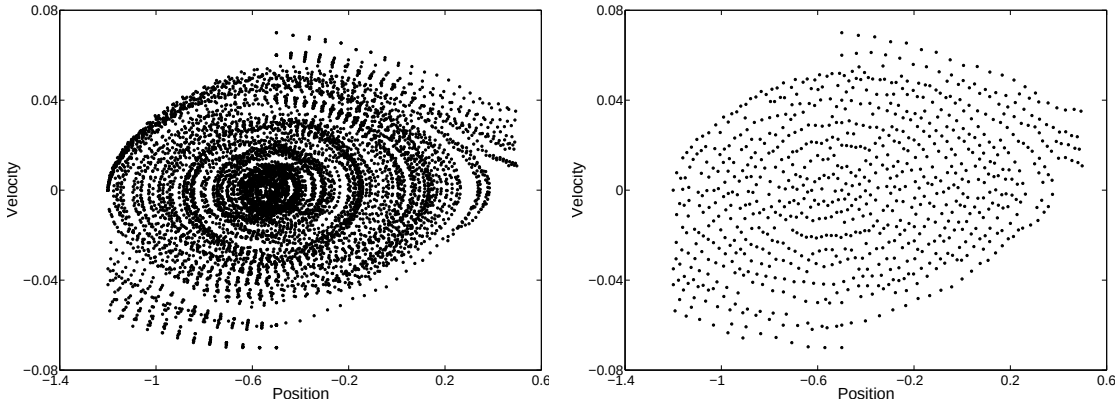
In this chapter, we present two approximation techniques for scaling to graphs (matrices) with a large number of vertices. The first technique factorizes the matrix into two smaller matrices. Eigenvectors or diffusion wavelets are then computed on the smaller matrices. The second technique uses a multilevel, parallel algorithm for computing eigenvectors. This algorithm benefits by performing the eigenvector computation on submatrices of the original matrix. Before describing both approximation techniques, we first discuss how sampling can be used as a preprocessing step to improve computational efficiency.

5.1 Sampling

Spectral bases are amenable to sparsification methods investigated in the kernel methods literature including low-rank approximation techniques as well as the Nyström interpolation method [108, 30]. The Nyström method allows for extrapolating functions on

sampled states to novel states. Subsampling the states using a greedy algorithm can greatly reduce the number of samples while still capturing the structure of the data manifold. The greedy algorithm is simple: starting with the null set, add samples to the subset that are **not** within a specified distance to any sample currently in the subset. A maximal subset is returned when no more samples can be added. The greedy procedure was originally proposed in a batch setting [93] and then later devised to work online [24, 35] where one must decided to discard or store samples as they are received. We used the batch greedy procedure for the experiments in this dissertation.

As an example of batch greedy subsampling, consider samples generated from a random policy in the mountain car task. Figure 5.1 shows the results of greedy subsampling on data from this domain. Clearly, the overall structure, but not the density, of the data still remains in the subsamples.



(a) 7,459 samples

(b) 846 subsamples

Figure 5.1. Greedy subsampling in the mountain car task.

It is natural to question how far the subsampling procedure can go before losing too much information. The answer is that it depends on the task. For example, in keeping with the mountain car task, Figure 5.2 shows the 3rd-5th eigenvectors of the graph Laplacian for two graphs: one with 50 samples and one with 500 samples. Clearly some information is lost, but the overall shape of the functions is very similar. There is a trade-off being made

here. More samples provides a higher resolution, but at the cost of increased computation. For RL, we want to find good policies. The "right" resolution to the subsampling problem is the minimal number of subsamples that allows for computing a good policy. We investigate this trade-off empirically.

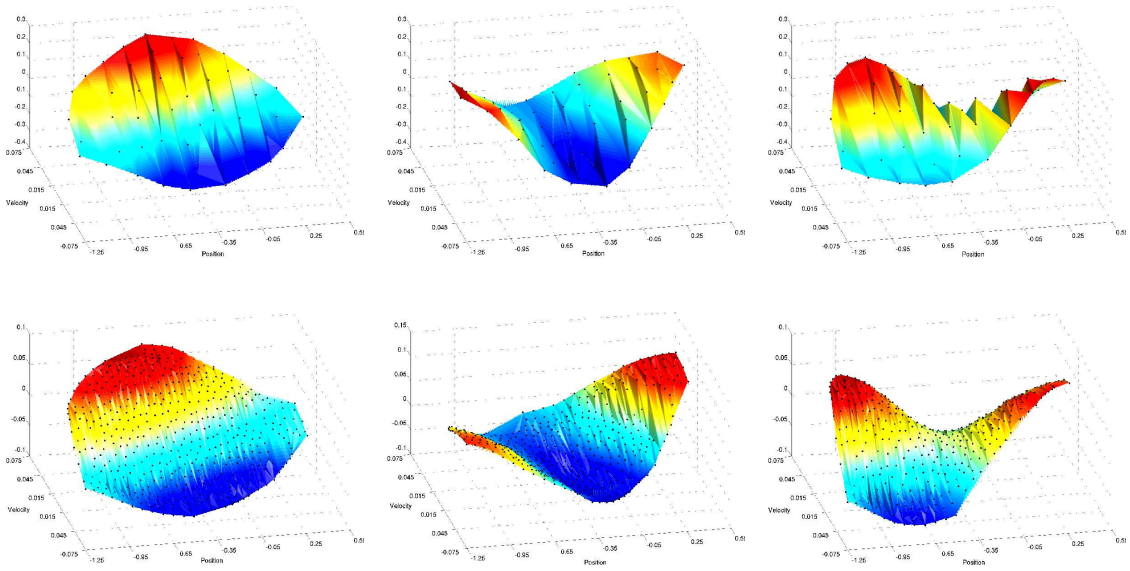


Figure 5.2. 3rd-5th Laplacian eigenvectors of two graphs from the mountain car domain. The top row is a graph with 50 vertices while the bottom row is a graph with 500 vertices. The left column is the 3rd eigenvector, middle column is the 4th eigenvector, and the right column is the 5th eigenvector.

Sampling helps to reduce the complexity of the graph-based basis construction problem, but it does not solve all our computational problems. Some domains naturally require many samples to provide adequate coverage of the state space. To deal with such large-scale problems, the next two sections of this chapter propose and evaluate algorithms for scaling basis construction to large graphs.

5.2 Matrix Factorization

Spectral basis functions, or PVFs, are not compact since they span the set of samples used to construct the graph. This raises a computational question of whether this approach

scales to large MDPs. We have explored a technique for making spectral bases compact using matrix factorization [46]. The main idea is that the random walk operator on the graph ($A \leftarrow D^{-1}W$) can be factored into two smaller stochastic matrices B and C such that the Kronecker product $B \otimes C \approx A$.[1] This procedure can be called recursively to further shrink the size of B and/or C. The Metropolis-Hastings algorithm is used to make B and C reversible, which ensures their eigendecompositions contain all real values. The result is the basis functions can be calculated from B and C rather than the original matrix A. This is a gain in terms of both speed and memory.

We present the Kronecker product approximation as a way to approximate eigenvector computation. However, the same ideas can also be used to approximate diffusion wavelets. In that case, diffusion wavelet construction occurs on the two smaller matrices B and C. The scaling and wavelet functions from the two smaller matrices can then be combined using the Kronecker product. We do not pursue this idea further, but point out that it would be interesting to explore different ways of combining functions from the two diffusion wavelet trees.

The following sections contain a description of the Kronecker product, Kronecker product approximation, a theoretical analysis describing when Kronecker product approximation works well, and some experiments.

5.2.1 Kronecker Product

The Kronecker product of a $r_B \times c_B$ matrix B and a $r_C \times c_C$ matrix C is equal to a matrix A of size $(r_B r_C) \times (c_B c_C)$ with block $A_{i,j} = B(i,j)C$. Thus, every (i,j) block of A is equal to the matrix C multiplied by the scalar $B(i,j)$. The equation is written $A = B \otimes C$.

[1] For ease of understanding, we use the symbols A, B, and C in Section 5.2 to represent the Kronecker product $A = B \otimes C$ or $A \approx B \otimes C$. Our use of the Kronecker product involves setting the matrix A to be the random walk operator $D^{-1}W$ associated with a graph. In previous chapters, the symbol $\mathcal{P}$ was used to represent $D^{-1}W$. We found using $\mathcal{P} = B \otimes C$ to be less readable than $A = B \otimes C$. Moreover, using $\mathcal{P} = B \otimes C$ gives the false impression that the Kronecker product only works with random walk operators.

The Kronecker product can be used to streamline many computations in numerical linear algebra, signal processing, and graph theory.

Assume B and C correspond to stochastic matrices associated with weighted, undirected graphs $G_B = (V_B, E_B, W_B)$ and $G_C = (V_C, E_C, W_C)$. The graphs can be represented as weight matrices W_B and W_C with strictly positive edge weights. Matrix B is then formed by dividing each row of W_B by the row sum (similarly for C). B and C are stochastic matrices representing random walks over their respective graphs. The eigenvalues and eigenvectors of B and C completely determine the eigenvalues and eigenvectors of $B \otimes C$.

Theorem 1 *Let B have eigenvectors x_i and eigenvalues λ_i for $1 \leq i \leq r_B$. Let C have eigenvectors y_j and eigenvalues μ_j for $1 \leq j \leq r_C$. Then matrix $B \otimes C$ has eigenvectors $x_i \otimes y_j$ and eigenvalues $\lambda_i \mu_j$.*

Proof: Consider $(B \otimes C)(x_i \otimes y_j)$ evaluated at vertex (v, w) where $v \in V_B$ and $w \in V_C$:

$$\begin{aligned}
(B \otimes C)(x_i \otimes y_j)(v, w) &= \sum_{(v,v_2) \in E_B} \sum_{(w,w_2) \in E_C} B(v, v_2) C(w, w_2) x_i(v_2) y_j(w_2) \\
&= \sum_{(v,v_2) \in E_B} B(v, v_2) x_i(v_2) \sum_{(w,w_2) \in E_C} C(w, w_2) y_j(w_2) \\
&= (\lambda_i x_i(v)) (\mu_j y_j(w)) \;=\; (\lambda_i \mu_j) (x_i(v) y_j(w)) \,. \square
\end{aligned}$$

This theorem is adapted from a more general version [8] that does not place constraints on the two matrices. Note this theorem also holds if B and C are normalized Laplacian matrices [20], but it does not hold for the combinatorial Laplacian. The Kronecker product is an important tool because the eigendecomposition of $A = B \otimes C$ can be accomplished by solving the smaller problems on B and C individually. The computational complexity of the eigendecomposition is reduced from $O(r_B^3 r_C^3)$ to $O(r_B^3 + r_C^3)$.

5.2.2 Kronecker Product Approximation

Given the computational benefits of the Kronecker factorization, it is natural to consider the problem of finding matrices B and C to approximate a matrix A. Pitsianis [84] studied this problem for *arbitrary* matrices. Specifically, given a matrix A, the problem is to minimize the function

$$f_A(B, C) = \|A - B \otimes C\|_F, \tag{5.1}$$

where $\|\cdot\|_F$ is the Frobenius norm. By reorganizing the rows and columns of A, the function f_A can be rewritten as:

$$f_A(B, C) = \|\tilde{A} - \text{vec}(B)\text{vec}(C)^T\|_F, \tag{5.2}$$

where the $\text{vec}(\cdot)$ operator takes a matrix and returns a vector by stacking the columns in order. The matrix $\tilde{A}$ is defined as:

$$\tilde{A} = \begin{bmatrix} \text{vec}(A_{1,1})^T \\ \vdots \\ \text{vec}(A_{r_B,1})^T \\ \vdots \\ \text{vec}(A_{1,c_B})^T \\ \vdots \\ \text{vec}(A_{r_B,c_B})^T \end{bmatrix} \in \mathbb{R}^{(r_B c_B)\times(r_C c_C)}. \tag{5.3}$$

Equation 5.2 shows the Kronecker product approximation problem is equivalent to a rank-one matrix problem. The solution to a rank-one matrix problem can be computed from the singular value decomposition (SVD) of $\tilde{A} = U\Sigma V^T$ [39]. The minimizing values

are vec$(B) = \sqrt{\sigma_1}u_1$ and vec$(C) = \sqrt{\sigma_1}v_1$ where u_1 and v_1 are the first columns of U and V and σ_1 is the largest singular value of $\tilde{A}$. This is done in time $O(r_B^2 r_C^2)$ since only the first singular value and singular vectors of the SVD are required.

Pitsianis [84] extended this idea to constrained optimization problems where the symmetry, orthogonality, or stochasticity of matrices B and C are preserved. We investigated the `kpa_markov` algorithm which finds *stochastic* matrices B and C that approximate a *stochastic* matrix A given as input. There are equality (row sums must sum to 1) and inequality (all values must be non-negative) constraints for this problem. The `kpa_markov` algorithm substitutes the equality constraints into the problem formulation and ignores the inequality constraints. One iteration of the algorithm proceeds by fixing C and updating B based on the derivative of $\|A - B \otimes C\|_F$; then matrix B is held constant and C is updated. Convergence is based on the change in the Frobenius norm. If the algorithm returned negative values, those entries were replaced with zeros and the rows were rescaled to sum to 1. More sophisticated algorithms (e.g. active set method) could be used to directly account for the inequality constraints if necessary.

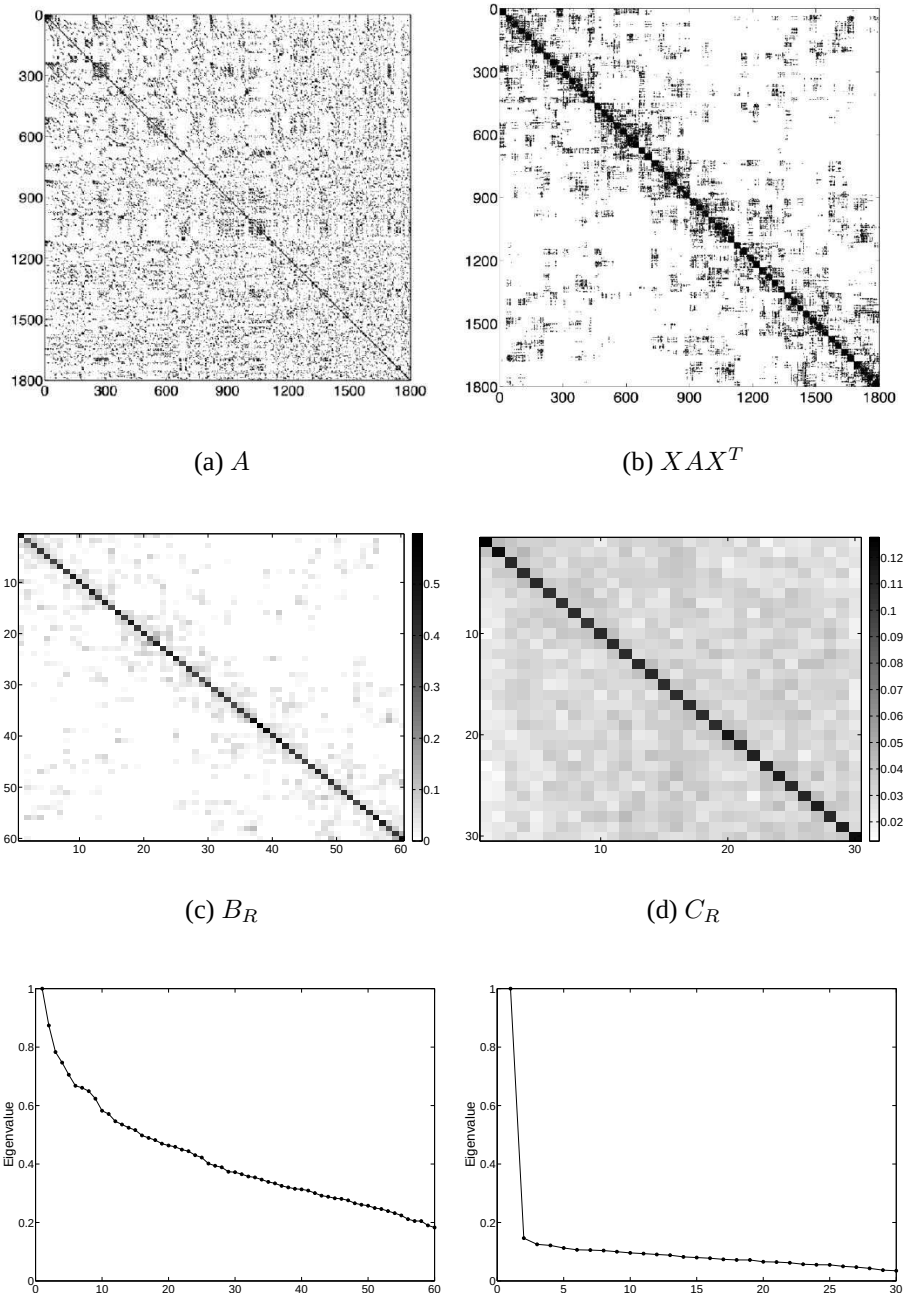
The Kronecker product has simple semantics when the matrices are stochastic. Matrix A is compacted into r_B clusters, each of size r_C. Matrix B contains transitions between clusters while matrix C contains transitions within a cluster. For the block structure of the Kronecker product to be most effective, similar states must be clustered. This can be achieved by reordering matrix A via XAX^T where X is a permutation matrix. A permutation matrix is a square $(0, 1)$ matrix that has exactly one 1 in each row and each column and 0's everywhere else. The problem of finding the optimal X to minimize $\|XAX^T - B \otimes C\|_F$ is NP-hard. However, there are several options for reordering matrices including graph partitioning and approximate minimum degree ordering. We used the graph partitioning program METIS [49] to determine X. METIS combines several heuristics for generating partitions, optimizing the balance of a partition versus the number of edges going across partitions. The algorithm first coarsens the graph, then partitions the smaller graph, and

finally uncoarsens and refines the partitions. METIS is a highly optimized program that partitions graphs with $O(10^6)$ vertices in a few seconds. Figure 5.3(a) shows an adjacency plot of a matrix A corresponding to a graph connecting 1800 sample states from the acrobot domain. Figure 5.3(b) is the same matrix but reordered according to METIS with 60 partitions. The reordered matrix is in a block structure more easily represented by the Kronecker decomposition.

The stochastic matrices B and C are not necessarily reversible. As such, their eigenvalues can be complex. To ensure all real values, we used the Metropolis-Hastings algorithm to convert B and C into reversible stochastic matrices B_R and C_R. The algorithm is described below where π is a stationary probability distribution.

$$B_R(i,j) = \begin{cases} B(i,j)\min\left(1, \dfrac{\pi(j)B(j,i)}{\pi(i)B(i,j)}\right) & \text{if } i \neq j \\ B(i,j) + \displaystyle\sum_k B(i,k) & \text{if } i = j \\ \quad \times \left(1 - \min\left(1, \dfrac{\pi(k)B(k,i)}{\pi(i)B(i,k)}\right)\right) & \end{cases}$$

This transformation was proven [13] to minimize the distance in an L_1 metric between the original matrix B and the space of reversible stochastic matrices with stationary distribution π. The power method [39] was used to determine the stationary distributions of B and C. Note these stationary distributions were unique in our experiments because B and C were both aperiodic and irreducible although the `kpa_markov` algorithm does not specifically maintain these properties. Figures 5.3(c) and 5.3(d) show grayscale images of the reversible stochastic matrices B_R and C_R that were computed by this algorithm to approximate the matrix in Figure 5.3(b). As these figures suggest, the Kronecker factorization is performing a type of state aggregation. The matrix B_R has the same structure as XAX^T, whereas C_R is close to a uniform block matrix except with more weight along the diagonal. The eigenvalues of B_R and C_R are displayed in Figures 5.3(e) and 5.3(f). The



(a) A (b) XAX^T

(c) B_R (d) C_R

(e) Eigenvalues of B_R (f) Eigenvalues of C_R

Figure 5.3. (a) Adjacency plot of an 1800×1800 matrix from the acrobot domain, (b) Matrix reordered using METIS, (c) 60×60 matrix B_R, (d) 30×30 matrix C_R, (e) Spectrum of B_R, and (f) Spectrum of C_R.

fact that C_R is close to a block matrix can be seen in the large gap between the first and second eigenvalues.

There is an added benefit of computing the stationary distributions. The eigendecomposition of B_R (and C_R) is less robust because the matrix is asymmetric. However, B_R is *similar* to a symmetric matrix $B_{R,\pi}$ by the equation $B_{R,\pi} = D_\pi^{0.5} B_R D_\pi^{-0.5}$ where D_π is a diagonal matrix with elements π. Matrices B_R and $B_{R,\pi}$ have identical eigenvalues and the eigenvectors of B_R can be computed by multiplying $D_\pi^{-0.5}$ by the eigenvectors of $B_{R,\pi}$.[2] Therefore, the decomposition should always be done on $B_{R,\pi}$.

It is far more economical to store the eigenvectors of B_R and C_R than those of A. For example, if 90 eigenvectors are used, then the eigenvectors of matrix A in Figure 5.3(a) consist of 162,000 values (1800×90). There are 3600 values (60×60) for B_R and 900 values (30×30) for C_R, yielding a compression ratio of 36 ($\frac{162{,}000}{3600+900}$).

There is a potentially subtle point to make regarding the space spanned by the matrices A, B_R, and C_R. Given the complete set of 1800 eigenvectors of A, one can represent *any* function over A's domain. The eigenvectors form a basis for all possible functions. One can also represent any function over A's domain given all 1800 possible combinations of the 60 eigenvectors of B_R and the 30 eigenvectors of C_R. In other words, the combination of B_R's and C_R's eigenvectors also forms a basis. This offers a useful perspective on the Kronecker product approximation. We know, based on the Dirichlet sum, that matrix A's eigenvectors are ordered (by increasing eigenvalue) in terms of smoothness. The smoothest eigenvectors capture the low frequency components for functions in A's domain. The Kronecker approximation method can be viewed as trying to similarly organize the space of functions in A's domain under the structural constraints imposed by the Kronecker product.

[2] In the experiments, we found it better to use the eigenvectors of $B_{R,\pi}$ as basis functions rather than B_R's eigenvectors. The eigenvectors of $B_{R,\pi}$ are orthonormal whereas the eigenvectors of B_R are orthonormal *with respect to* π (i.e. the weighted inner product $\langle \cdot, \cdot \rangle_\pi$ of two different eigenvectors of B_R is 0).

The 1800 possible combinations of the 60 eigenvectors of B_R and the 30 eigenvectors of C_R provide a basis for matrix A's domain. The experiments in this chapter generate K "combined" eigenvectors from this set of 1800 by selecting those with the smallest value of the product of B_R's and C_R's eigenvalues (with ties broken using the sum of eigenvalues). We point out that it is also possible to use the eigenvectors of B_R and C_R as a dictionary and to select the best set of functions for a particular task. We study basis selection algorithms in Chapter 6.

5.2.3 Extensions

We consider two extensions to the Kronecker product framework. The first extension is to expand the number of matrices used in the Kronecker product decomposition from $p = 2$ to $p \geq 2$ matrices. A decomposition into $p \geq 2$ matrices would provide greater compression. The second extension is to shift from a rank-1 Kronecker product approximation to a rank-r approximation which would increase the Kronecker product's expressiveness.

The Kronecker product approximation works by decomposing a matrix A into matrices B and C such that $\|A - B \otimes C\|_F$ is minimized. The matrices B and C are then, in effect, used as a proxy for A. The advantage of the approximation is that B and C are smaller than A, thereby decreasing the computational complexity of any algorithm operating on A. To further decrease the amount of computation, the matrices B and C can themselves be decomposed using the same Kronecker product framework. More generally, we can minimize the following function:

$$f_A(B_1, B_2, \ldots, B_p) = \|A - B_1 \otimes B_2 \otimes \ldots \otimes B_p\|_F$$

where $p \geq 2$ terms are used.

Algorithmically, this can be accomplished by using the algorithm described in the previous section recursively. There is however another interesting way to solve this problem. Recall in Equation 5.2 we showed the function $f_A(B, C)$ could be rewritten to have the

form $f_A(B, C) = \|\tilde{A} - \text{vec}(B)\text{vec}(C)^T\|_F$. The reorganized matrix $\tilde{A}$ is approximated using the outer product of vectors vec(B) and vec(C). The outer product, or *tensor product*, of two vectors results in a matrix. Now, consider the tensor product of the p vectors: vec$(B_1) \circ$ vec$(B_2) \circ \ldots \circ$ vec(B_p). This results in a p dimensional tensor. For example, a $p = 3$ dimensional tensor can be viewed as a three dimensional rectangle. The shift from vectors and matrices to tensors corresponds to the extension from linear algebra to multilinear algebra [72].

This insight leads to an elegant way of extending the Kronecker product approximation problem. The matrix A can be rearranged into a p dimensional tensor $R_p(A)$. When $p = 2$, the rearrangement operator takes the form of Equation 5.3 (i.e. $R_2(A) = \tilde{A}$). Given the p dimensional tensor $R_p(A)$, the so called *higher order singular value decomposition* (HOSVD) [28] can be used to find the values of vec(B_1), vec$(B_2), \ldots,$ vec(B_p). It turns out the HOSVD does not quite produce the *optimal* rank-1 p^{th} order tensor approximation of $R_p(A)$, but it does provide a *good* rank-1 approximation [29, 58].[3] This is directly analogous to the two dimensional case where the optimal rank-1 matrix approximation is computed using the traditional SVD. To complete this description, we need to specify the rearrangement operator $R_p(\cdot)$. Assume $B_1, B_2, \ldots, B_p$ are square matrices of size $r_{B_1}, r_{B_2}, \ldots, r_{B_p}$. Thus, the square matrix A is of size $(r_{B_1} r_{B_2} \ldots r_{B_p})$. For the purpose of defining $R_p(\cdot)$, we say A is made up of $r_{B_1}^2$ blocks, each block of size $(r_{B_2} r_{B_3} \ldots r_{B_p}) \times (r_{B_2} r_{B_3} \ldots r_{B_p})$. The rearrangement operator $R_p(A)$ can be defined recursively as vectorizing the $r_{B_1}^2$ blocks of matrix A and applying $R_{p-1}(\cdot)$ to each block.

Extending the Kronecker product approximation framework recursively results in even smaller matrices. This greatly speeds up eigendecomposition or diffusion wavelet construction, but it imposes even more block structure on the problem. Details to the original problem may be entirely lost by enforcing such strong structure on an unstructured matrix.

[3] de Lathauwer et al. [28] provide a more complicated algorithm than the truncated HOSVD for computing an optimal rank-1 p^{th} order tensor approximation. For practical reasons, we recommend using the HOSVD.

The second extension to the Kronecker product framework is to perform a rank-r approximation instead of a rank-1 approximation. We showed that the Kronecker product approximation problem of minimizing $\|A - B \otimes C\|_F$ can be rearranged to have the form $\|\tilde{A} - \text{vec}(B)\text{vec}(C)^T\|_F$. The outer product $\text{vec}(B)\text{vec}(C)^T$ is necessarily a rank-1 matrix. The Eckart-Young theorem [32] proves that the optimal rank-r approximation of a matrix can be found using the singular value decomposition and keeping the r largest singular values (and zeroing out all other singular values). For the Kronecker product approximation problem, $\text{vec}(B)$ and $\text{vec}(C)$ are found by computing the largest singular value and singular vectors of $\tilde{A}$.

It is natural to consider whether this framework can be extended from a rank-1 approximation to a more expressive rank-r approximation. The Kronecker product approximation problem *can* be extended from a rank-1 to a rank-r problem by altering the original problem as follows:

$$f_A(\,\{B_i\}_{i=1}^r,\,\{C_i\}_{i=1}^r\,) = \|A - \sum_{i=1}^{r} B_i \otimes C_i\|_F.$$

Not surprisingly, this equation can be reordered to yield $\|\tilde{A} - \sum_{i=1}^r \text{vec}(B_i)\text{vec}(C_i)^T\|_F$. The optimal values of $\{\text{vec}(B_i)\}_{i=1}^r$ and $\{\text{vec}(C_i)\}_{i=1}^r$ can be found by computing the r largest singular values and vectors of $\tilde{A}$.

Clearly the sum of r Kronecker products $\sum_{i=1}^r \text{vec}(B_i)\text{vec}(C_i)^T$ yields a better approximation to $\tilde{A}$ than just using a single Kronecker product. Unfortunately, we cannot use this property to help with the original eigendecomposition problem. Theorem 1 proved that if $A = B \otimes C$, the eigenvectors and eigenvalues of A are fully determined by the eigenvectors and eigenvalues of B and C. This theorem cannot be extended to the case when $A = \sum_{i=1}^r B_i \otimes C_i$. In other words, knowing the eigenvectors and eigenvalues of $\{B_i\}_{i=1}^r$ and $\{C_i\}_{i=1}^r$ does not uniquely determine the eigenvalues and eigenvectors of A.

5.2.4 Theoretical Analysis

This analysis attempts to shed some light on when $B \otimes C$ is useful for approximating A.[4] More specifically, we are concerned with whether the space spanned by the top m eigenvectors of $B \otimes C$ is "close" to the space spanned by the top m eigenvectors of A. Perturbation theory can be used to address this question because the random walk operator A is self-adjoint (with respect to the *invariant distribution* of the random walk) on an inner product space; therefore, theoretical results concerning A's spectrum apply. We assume matrices B and C are computed according to the constrained Kronecker product approximation algorithm discussed in the previous section. The following notation is used in the theorem and proof:

- $E = A - B \otimes C$
- X is a matrix whose columns are the top m eigenvectors of A
- α_1 is the set of top m eigenvalues of A
- α_2 includes all eigenvalues of A except those in α_1
- d is the eigengap between α_1 and α_2, i.e., $d = \min_{\lambda_i \in \alpha_1, \lambda_j \in \alpha_2} |\lambda_i - \lambda_j|$
- Y is a matrix whose columns are the top m eigenvectors of $B \otimes C$
- $\tilde{\alpha_1}$ is the set of top m eigenvalues of $B \otimes C$
- $\tilde{\alpha_2}$ includes all eigenvalues of $B \otimes C$ except those in $\tilde{\alpha_1}$
- $\tilde{d}$ is the eigengap between α_1 and $\tilde{\alpha_2}$
- $\mathcal{X}$ is the subspace spanned by X
- $\mathcal{Y}$ is the subspace spanned by Y
- P is the orthogonal projection onto $\mathcal{X}$
- Q is the orthogonal projection onto $\mathcal{Y}$.

Theorem 2 *Assuming B and C are defined as above based on the SVD of $\tilde{A}$ and if $\|E\| \leq 2\varepsilon d/(\pi + 2\varepsilon)$, then the distance between the space spanned by top m eigenvectors of A and the space spanned by the top m eigenvectors of $B \otimes C$ is at most ε.*

[4] We gratefully acknowledge the help of Chang Wang with the analysis in this section.

Proof: The Kronecker factorization uses the top m eigenvectors of $B \otimes C$ to approximate the top m eigenvectors of A (e.g. use Y to approximate X). The difference between $\mathcal{X}$ and $\mathcal{Y}$ is defined $\|Q - P\|$. [$\mathbf{S}_1$]

It can be shown that if A and E are bounded self-adjoint operators on a separable Hilbert space, then the spectrum of A+E is in the closed $\|E\|$-neighborhood of the spectrum of A [54]. The authors also prove the inequality $\|Q^{\perp}P\| \leq \pi\|E\|/2\tilde{d}$. [$\mathbf{S}_2$]

Matrix A has an isolated part α_1 of the spectrum separated from its remainder α_2 by gap d. To guarantee A+E also has separated spectral components, we need to assume $\|E\| < d/2$. Making this assumption, [$\mathbf{S}_2$] can be rewritten $\|Q^{\perp}P\| \leq \pi\|E\|/2(d - \|E\|)$. [$\mathbf{S}_3$]

Interchanging the roles of α_1 and α_2, we have the analogous inequality: $\|QP^{\perp}\| \leq \pi\|E\|/2(d - \|E\|)$. [$\mathbf{S}_4$] Since $\|Q - P\| = \max\{\|Q^{\perp}P\|, \|QP^{\perp}\|\}$ [$\mathbf{S}_5$], the overall inequality can be written $\|Q - P\| \leq \pi\|E\|/2(d - \|E\|)$. [$\mathbf{S}_6$]

Step [$\mathbf{S}_6$] implies that if $\|E\| \leq 2\varepsilon d/(\pi + 2\varepsilon)$, then $\|Q - P\| \leq \varepsilon$. [$\mathbf{S}_7$] $\square$

The two important factors involved in this theorem are $\|E\|$ and the eigengap of A. If $\|E\|$ is small, then the space spanned by the top m eigenvectors of $B \otimes C$ approximates the space spanned by the top m eigenvectors of A well. Also, for a given value of $\|E\|$, the larger the eigengap the better the approximation.

5.2.5 Experiments

The experiments were conducted using the mountain car domain and the acrobot task. The experimental setup follows the RPI algorithm described in Figure 4.6. A comparison was done using basis functions derived from the matrix random walk operator $A = D^{-1}W$ versus basis functions from the factorized matrices B_R and C_R. Samples were generated using a random policy. For the mountain car task, each episode began at the bottom of the hill but with a different initial velocity (see the Appendix for details). Episodes lasted for at most 50 steps or until the goal was reached. We used a slightly more involved sample

generation procedure for acrobot. Since random policies can take a very long time to reach the goal, we only used samples from episodes that reached the goal in under 800 time steps. We found this provided coverage of samples over the state-action space using a random policy while minimizing redundancy in the sample set.

After generating the samples, we used the greedy subsampling method described in Section 5.1 to arrive at a representative set of data points. Graphs were then built by connecting each subsampled state to its k nearest neighbors and edge weights were assigned using a *weighted* Euclidean distance metric. A weighted Euclidean distance metric was used as opposed to an unweighted metric to make the state space dimensions have more similar ranges. These parameters are given in the first three rows of Table 5.1. There is one important exception for graph construction in acrobot. The joint angles θ_1 and θ_2 range from 0 to 2π; therefore, arc length is the appropriate distance metric to ensure values slightly greater than 0 are "close" to values slightly less than 2π. However, the fast nearest neighbor code[5] that was used to generate graphs required a Euclidean distance metric. To approximate arc length using Euclidean distance, angle θ_i was mapped to a tuple [$\sin(\theta_i)$, $\cos(\theta_i)$] for $i = \{1, 2\}$. This approximation works very well if two angles are similar (e.g. within 30 degrees of each other) and becomes worse as the angles are further apart. Next, matrices A, B_R, and C_R were computed using the Kronecker factorization algorithm previously discussed. By fixing the size of C_R, the size of B_R is automatically determined by $|B_R| = \frac{|A|}{|C_R|}$. To ensure $\frac{|A|}{|C_R|}$ is an integer, we simply adjust the number of elements in A. After the greedy subsampling procedure is run, extra samples can either be added or removed to the subset. The last four rows of Table 5.1 show the sizes of B_R and C_R, the number of eigenvectors used, and the compression ratios achieved by storing the compact basis functions. Notice we used more eigenvectors from $B_R \otimes C_R$ than we did from A. These number were determined empirically to work well. Results were worse when more

[5]We thank Dr. Christian Merkwirth who kindly sent us the source code for his fast nearest neighbor package, ATRIA [71].

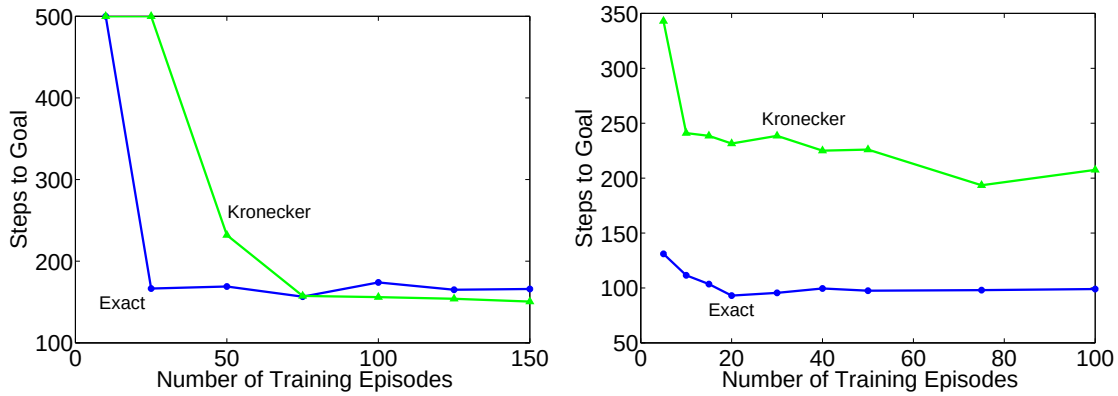
	Mountain Car	Acrobot
γ	0.99	1.0
k	25	15
σ	0.2	0.5
Weight	$[1, 24]$	$[1.0, 1.0, 0.7, 0.7, 0.3, 0.12]$
	$[x, \dot{x}]$	$\left[\sin(\theta_1), \cos(\theta_1), \sin(\theta_2), \cos(\theta_2), \dot{\theta}_1, \dot{\theta}_2\right]$
Size A	$\sim$1000	$\sim$3000
Eigenvectors of A	20	25
Size B_R	$\sim$100	$\sim$200
Size C_R	10	15
Eigenvectors of $B_R \otimes C_R$	50	50
Compression Factor	$\sim$9.8 $\times$	$\sim$14.7 $\times$

Table 5.1. Parameters for the experiments.

eigenvectors were used from A and fewer eigenvectors were used from $B_R \otimes C_R$. Also, note that the eigenvectors were used to represent action-value functions. This was done by using the eigenvectors separately for each discrete action as described in Section 3.2. Thus, when 20 eigenvectors of A were computed for mountain car, this means 60 basis functions (20 eigenvectors $\times$ 3 discrete actions) were used to approximate the action-value function.

The goal of our experiments was to compare the effectiveness of the basis functions derived from matrix A with the basis functions derived from matrices B_R and C_R. Thirty separate trials were run for each domain. Within each trial, we varied the number of training episodes. Given a set of training data, we ran the greedy subsampling procedure, formed a k nearest neighbor graph, and computed the matrices A, B_R, and C_R. The eigenvectors of A form one set of basis functions while the eigenvectors of B_R and C_R form another set of basis functions. We refer to the basis functions as either the "exact" bases (those from A) or the "Kronecker" bases (those from B_R and C_R). The LSPI algorithm was run twice, once with the exact bases and once with the Kronecker bases. The learned policies from these two runs were evaluated starting from each tasks' typical initial state, $[x, \dot{x}] = [-0.5, 0]$ for mountain car and $\left[\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2\right] = [0, 0, 0, 0]$ for acrobot. We recorded the number of steps each policy took to reach the goal. The test was terminated for both domains if the policy

did not reach the goal in 500 steps. The median test results over the 30 runs are plotted in Figure 5.4. The median is a more appropriate measure of performance than the mean since some of the 30 runs resulted in policies that never reached the goal.



(a) Mountain Car (b) Acrobot

Figure 5.4. Median performance over the 30 runs using the RPI algorithm and the parameters described in Table 5.1. The basis functions are either derived from matrix A (Exact) or from matrices B_R and C_R (Kronecker).

The results from mountain car show that policies performed similarly whether using the exact basis functions or the Kronecker basis functions. The only difference occurs when the amount of training data is small (< 50 episodes). In that case, the exact basis functions resulted in better policies.

There was a significant difference in performance for the acrobot task. The policies learned using the exact basis functions performed much better than those learned using the Kronecker basis functions. With only 15 episodes of training data, the policies learned using the exact bases were able to reach the goal in $\sim$100 time steps. This is relatively close to an optimal policy which reaches the goal in $\sim$75 time steps. We experimented with different parameters for the Kronecker product method. We ranged the number of eigenvectors of $B_R \otimes C_R$ and the size of the matrix C_R. These changes did not result in policies that could reach the goal in under $\sim$200 time steps (over the median of the 30 trials). We did find that changing the distance function used to create the graph had a substantial im-

pact on the results. For example, in previous work [46], we used a weighted Euclidean distance metric with the dimensions scaled as $\left[\sin(\theta_1), \cos(\theta_1), \sin(\theta_2), \cos(\theta_2), \dot{\theta}_1, \dot{\theta}_2\right] = [1, 1, 1, 1, 0.5, 0.3]$. This distance metric puts more emphasis on the angular velocities than the distance metric used in the experiments above (see Table 5.1). Using this distance function, $|C_R|$ =30, and 90 eigenvectors of $B_R \otimes C_R$, the performance of exact and Kronecker bases became more similar. Both sets of basis functions resulted in policies that reach the goal in ∼150 time steps. This is a noticeable decline in performance for the exact basis functions (an increase from 100 to 150 steps) and an improvement for the Kronecker basis functions (a decrease from 200 to 150 steps). These results, shown in Figure 5.5, indicate the Kronecker product method can depend on the particular graph and distance function being used. This makes using the Kronecker product technique more challenging.

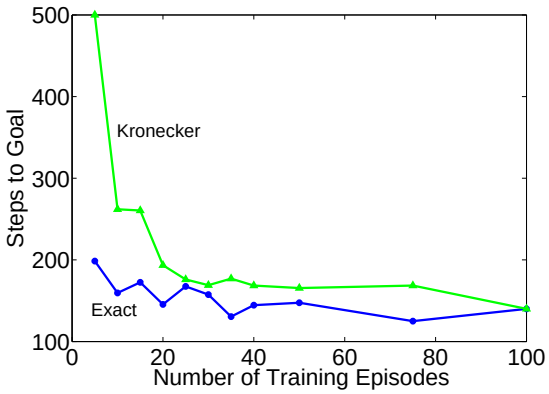


Figure 5.5. Median performance over 30 trials using the RPI algorithm on acrobot. Graphs were constructed using a different distance function than was used for the acrobot plot in Figure 5.4.

5.3 Multilevel Eigenvector Approximation

In this section, we present an alternative way to compute approximate eigenvectors. The Automated Multilevel Substructuring (AMLS) algorithm [9] was recently introduced as a way to scale up eigenvector computation to very large-scale problems. We describe the algorithm and then prove it is applicable to computing eigenvectors of the graph Laplacian.

The AMLS and the Kronecker product method are compared and experimental results are presented.

5.3.1 Automated Multilevel Substructuring

Bennighof and Lehoucq [9] developed the Automated Multilevel Substructuring (AMLS) algorithm to solve large, sparse symmetric eigenvalue problems. They reported computing thousands of eigenpairs on a matrix with millions of rows using a commodity computer and doing so orders of magnitude faster than current state-of-the-art algorithms. AMLS is well-suited to problems where a large number of eigenvalues are to be computed [110].

The algorithm can be viewed as a multilevel extension of the component mode synthesis (CMS) [43] technique from structural dynamics. AMLS is based on domain decomposition where *a larger problem is divided into smaller subproblems whose solutions are found and then used as a subspace for approximately solving the larger problem.* The decomposition of a larger problem into smaller subproblems is carried out recursively, thus giving AMLS its "multilevel" nature. Our description of AMLS follows along the algebraic version of Bekas and Saad [4] instead of the original domain decomposition viewpoint [9].

We present a one-level version of the AMLS algorithm. It is easy to extend the algorithm recursively to multiple levels. Since the recursion involves *independent* subproblems, the algorithm can easily be parallelized. We have written both a sequential and parallel implementation of the AMLS algorithm in Matlab. The parallel version has been successfully executed on matrices with $O(10^6)$ rows using a computer cluster. However, we emphasize that AMLS does not require expensive computing resources and in fact greatly reduces the computational workload compared to the best existing algorithms. Bennighof and Lehoucq [9] point out that AMLS has been used commercially by auto manufacturers to compute twice as many eigenpairs "with commodity workstations in an order of magnitude less computing time than the standard [Lanczos] approach on a CRAY SV1 supercomputer."

Let $A \in \mathbb{R}^{n \times n}$ be the matrix given as input to AMLS. The desired outputs are approximations of the smallest K eigenvalues of A and their corresponding eigenvectors. We assume the rows and columns of A have been reordered to decompose the matrix into two independent blocks that are connected by a (ideally small) block separating them. This can be accomplished very quickly using sparse matrix partitioning methods. We used the *nested dissection* algorithm (`onmetis`) in the METIS software package [49]. Figure 5.6 shows how nested dissection reorders a matrix associated with a 50×50 two-dimensional grid ($n = 2500$). The plot on the left shows the original matrix ordering where blue dots correspond to nonzero matrix entries. The plot on the right shows the same matrix after reordering the rows and columns using nested dissection. The red lines on the right-hand plot delineate the matrix blocks (with the small separating block ordered last). Notice the same general structure is recursively repeated within both of the large independent blocks.

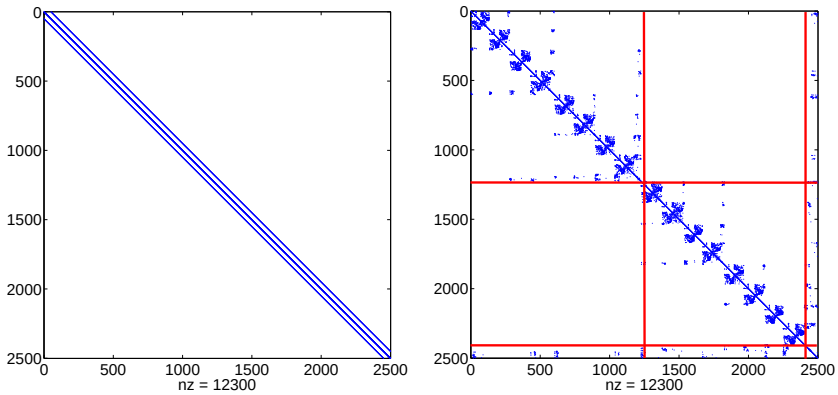


Figure 5.6. Connectivity plot of a matrix for a 50×50 grid graph (left) and the same matrix after the rows and columns have been reordered using nested dissection (right). The red lines are just shown to emphasize the reordered matrix consists of two (large) independent blocks followed by a small block separating them.

Given the nested dissection ordering, the matrix A has the form:

$$A = \begin{bmatrix} B_1 & 0 & E_1 \\ 0 & B_2 & E_2 \\ E_1^T & E_2^T & C \end{bmatrix} = \begin{bmatrix} B & E \\ E^T & C \end{bmatrix},$$

where matrix B is block diagonal. The eigenvalue problem $A\phi = \lambda\phi$ can be similarly written:

$$A\phi = \begin{bmatrix} B_1 & 0 & E_1 \\ 0 & B_2 & E_2 \\ E_1^T & E_2^T & C \end{bmatrix} \begin{bmatrix} \phi^{B_1} \\ \phi^{B_2} \\ \phi^C \end{bmatrix} = \lambda \begin{bmatrix} \phi^{B_1} \\ \phi^{B_2} \\ \phi^C \end{bmatrix} = \lambda \begin{bmatrix} \phi^B \\ \phi^C \end{bmatrix} = \lambda\phi.$$

The block Gaussian eliminator for A is the matrix

$$U = \begin{bmatrix} I & -B^{-1}E \\ 0 & I \end{bmatrix}.$$

This means $U^T AU$ results in a block diagonal matrix

$$U^T AU = \begin{bmatrix} B & 0 \\ 0 & S \end{bmatrix},$$

where $S = C - E^T B^{-1} E$ is the *Schur complement*. Instead of solving the eigenvalue problem $A\phi = \lambda\phi$, one can solve the equivalent eigenvalue problem $U^T AU\tilde{\phi} = \lambda U^T U\tilde{\phi}$ which has the form

$$(U^T AU)\tilde{\phi} = \begin{bmatrix} B & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} \tilde{\phi}^B \\ \tilde{\phi}^S \end{bmatrix} = \lambda \begin{bmatrix} I & -B^{-1}E \\ -E^T B^{-1} & M_S \end{bmatrix} \begin{bmatrix} \tilde{\phi}^B \\ \tilde{\phi}^S \end{bmatrix}, \qquad (5.4)$$

where $M_S = (I + E^T B^{-2} E)$. Notice the eigenvalues are the same whether solving $A\phi = \lambda\phi$ or $U^T A U\tilde{\phi} = \lambda U^T U\tilde{\phi}$. The eigenvectors are related by the simple transformation $\phi = U\tilde{\phi}$ or, more explicitly:

$$\phi = \begin{bmatrix} \phi^B \\ \phi^C \end{bmatrix} = U\tilde{\phi} = \begin{bmatrix} I & -B^{-1}E \\ 0 & I \end{bmatrix} \begin{bmatrix} \tilde{\phi}^B \\ \tilde{\phi}^S \end{bmatrix} = \begin{bmatrix} \tilde{\phi}^B - B^{-1}E\tilde{\phi}^S \\ \tilde{\phi}^S \end{bmatrix}.$$

Everything until this point has been done exactly. So solving the eigenvalue problem $U^T A U\tilde{\phi} = \lambda U^T U\tilde{\phi}$ results in eigenpairs that can be transformed into the exact eigenpairs of $A\phi = \lambda\phi$. Here we introduce an approximation. Instead of solving $U^T A U\tilde{\phi} = \lambda U^T U\tilde{\phi}$, we solve a slightly different eigenvalue problem in which the off-diagonal elements of $U^T U$ are ignored. By ignoring the off-diagonal blocks, we mean:

$$\begin{bmatrix} B & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} \tilde{\phi}^B \\ \tilde{\phi}^S \end{bmatrix} = \lambda \begin{bmatrix} I & \cancelto{0}{-B^{-1}E} \\ \cancelto{0}{-E^T B^{-1}} & M_S \end{bmatrix} \begin{bmatrix} \tilde{\phi}^B \\ \tilde{\phi}^S \end{bmatrix}.$$

With the off-diagonal blocks ignored, the problem decomposes into three *separate* eigenvalue problems: $B_1 v^{B_1} = \mu^{B_1} v^{B_1}$, $B_2 v^{B_2} = \mu^{B_2} v^{B_2}$, and $S v^S = \mu^S M_S v^S$. This is the essence of AMLS; the smaller problems are solved instead of directly tackling the larger problem.

We compute the n_1 eigenvectors $\{v_i^{B_1}\}_{i=1}^{n_1}$ associated with the smallest eigenvalues of B_1, the n_2 eigenvectors $\{v_i^{B_2}\}_{i=1}^{n_2}$ associated with the smallest eigenvalues of B_2, and the n_S eigenvectors $\{v_i^S\}_{i=1}^{n_S}$ associated with the smallest generalized eigenvalues of (S, M_S). If we are required to compute K eigenpairs of matrix A, then $(n_1 + n_2 + n_S) \geq K$. Computing more eigenvectors (i.e. larger $(n_1 + n_2 + n_S)$) results in more computation but a better approximation.[6] Once the eigenvectors are computed, they are used as a basis

[6]Bounds on the quality of the approximation are derived in [34].

for approximating the full problem $U^T A U\tilde{\phi} = \lambda U^T U\tilde{\phi}$. Specifically, a basis matrix Z is formed where:

$$Z = \begin{bmatrix} \{v_i^{B_1}\}_{i=1}^{n_1} & 0 & 0 \\ 0 & \{v_i^{B_2}\}_{i=1}^{n_2} & 0 \\ 0 & 0 & \{v_i^{S}\}_{i=1}^{n_S} \end{bmatrix},$$

and then the full problem is projected as $(Z^T U^T A U Z)z = \eta(Z^T U^T U Z)z$. Projecting the problem down to a subspace defined by Z is known as the Rayleigh-Ritz method. K eigenpairs associated with the smallest eigenvalues of the projected problem are then computed. This results in a set of eigenvalues $\{\eta_i\}_{i=1}^K$ and eigenvectors $\{z_i\}_{i=1}^K$. The AMLS algorithm outputs:

1. $\{\eta_i\}_{i=1}^K$ as an approximation to A's eigenvalues $\{\lambda_i\}_{i=1}^K$, and
2. $UZ\{z_i\}_{i=1}^K$ as an approximation to A's eigenvectors $\{\phi_i\}_{i=1}^K$.

Pseudocode for the AMLS algorithm is shown in Algorithm 4. Aside from the matrix A and the desired number of eigenpairs K, the other inputs to the algorithm are the number of eigenvectors to compute for the subproblems (n_1, n_2, and n_S). In our implementation, we used a simple heuristic to automatically select these numbers. First, we determined how many total eigenvectors to compute. This was done by multiplying K by a number greater than 1 (e.g. 1.67). Then we set n_1, n_2, and n_S to sum to that number and be in equal proportion to the size of matrices B_1, B_2, and C. As an example, say we needed to compute $K = 600$ eigenvectors of A and assume matrix A has dimension $20,000$, B_1 has dimension $10,000$, B_2 has dimension 9600, and C has dimension 400. First, for a factor of 1.67, we determine we need to compute $K \times 1.67 = 1000$ eigenvectors for all three subproblems. Then we apportion those eigenvectors as $n_1 = (\frac{10{,}000}{20{,}000}) \times 1000 = 500$, $n_2 = (\frac{9600}{20{,}000}) \times 1000 = 480$, and $n_S = (\frac{400}{20{,}000}) \times 1000 = 20$. Lastly, we adjusted n_1, n_2,

Algorithm 4: AMLS for one level of subproblems

Input: $A \in \mathbb{R}^{n \times n}$

K, desired number of eigenpairs of A to be computed

n_1, n_2, n_S, number of eigenvectors to compute for the eigendecomposition of B_1, B_2, and (S, M_S) respectively

Output: K approximate eigenpairs of A

// All eigendecompositions below compute the smallest eigenvalues

1. If necessary, reorder A using nested dissection:
$$A = \begin{bmatrix} B_1 & 0 & E_1 \\ 0 & B_2 & E_2 \\ E_1^T & E_2^T & C \end{bmatrix} = \begin{bmatrix} B & E \\ E^T & C \end{bmatrix}$$
2. Compute Cholesky factorization of B:
$R^T R = B$ where R is lower triangular
3. Compute the Schur complement S:
$S = C - E^T B^{-1} E$ *(using R)*
4. Define block Gaussian eliminator U:
$$U = \begin{bmatrix} I & -B^{-1}E \\ 0 & I \end{bmatrix}$$
5. Compute $M_S = I + \left(E^T B^{-1}\right)\left(B^{-1}E\right)$ *(using R)*
6. Compute eigenvectors of subproblems:
6a. Compute n_1 eigenvectors of B_1: $\{v_i^{B_1}\}_{i=1}^{n_1}$
6b. Compute n_2 eigenvectors of B_2: $\{v_i^{B_2}\}_{i=1}^{n_2}$
6c. Compute n_S generalized eigenvectors of (S, M_S): $\{v_i^{S}\}_{i=1}^{n_S}$
7. Define the matrix Z:
$$Z = \begin{bmatrix} \{v_i^{B_1}\}_{i=1}^{n_1} & 0 & 0 \\ 0 & \{v_i^{B_2}\}_{i=1}^{n_2} & 0 \\ 0 & 0 & \{v_i^{S}\}_{i=1}^{n_S} \end{bmatrix}$$
8. Compute K eigenpairs of $\left(Z^T U^T A U Z\right) z = \eta \left(Z^T U^T U Z\right) z$
9. Output the eigenvalues $\{\eta_i\}_{i=1}^{K}$ and eigenvectors $UZ\{z_i\}_{i=1}^{K}$

and n_S to not be less than a minimum threshold (e.g. 200). An alternative approach could select n_1, n_2, and n_S based on a desired level of accuracy [34].

The computational complexity of the AMLS algorithm depends on the structure of the input matrix, the number of levels in the nested dissection (i.e., how many levels of recursion), and the number of eigenpairs computed at each level. As such, it is not as easily characterized as the Kronecker product method. Gao et al. [38] provide empirical evidence of AMLS's run time for four test problems while varying the number of levels and the number of eigenpairs. Their analysis shows that, as expected, AMLS outperforms a common eigensolver (using a shift-and-invert Lanczos method) as the desired number of

eigenpairs increases. Thus, AMLS is ideally suited for our purpose of generating as many eigenvectors as possible from large, sparse Laplacian matrices.

5.3.2 AMLS and the Graph Laplacian

Bekas and Saad [4] describe the AMLS algorithm and some extensions. Their description of AMLS requires the input matrix to be symmetric positive definite (SPD). This would seem to be a problem for the graph Laplacian because it is positive *semi*definite. However, from our description of Algorithm 4 in the previous section, the only time the SPD property is used is in the Cholesky factorization. Moreover, the Cholesky factorization is only computed on a (permuted) principal submatrix of the input matrix and never on the entire input matrix itself. This means the graph Laplacian can be used with AMLS because *any* (permuted) principal submatrix of the Laplacian is SPD. The proof of this fact is shown in the lemma below.

Lemma 3 *Any principal submatrix of a graph Laplacian associated with a weighted, undirected, and connected graph is symmetric positive definite (SPD).*

Proof: Let L be a (normalized or combinatorial) graph Laplacian matrix associated with a weighted, undirected, and connected graph $G = (V, E, W)$ with vertex set V, edge set E, and edge weights W. By definition, L is a symmetric, positive semidefinite matrix. Given *any* permutation matrix X, let $L_X = XLX^T$ (i.e. L_X is a reordering of the rows and columns of L). L_X is also symmetric positive definite since reordering the rows and columns of L does not change these properties. Partition the matrix as:

$$L_X = \begin{bmatrix} A & B \\ B^T & C \end{bmatrix}$$

where the size of matrix A is $k \times k$ where $1 \leq k < |V|$. Matrix A is the leading principal submatrix of L_X. Since L_X is symmetric, A is also symmetric.

Now, let V_A be a restriction of the vertex set V to the k elements contained in A. Likewise, let V_C be a restriction of V to the $(|V| - k)$ elements contained in C. Let E_A contain all the edges in the set E connecting vertices in the set V_A. With these definitions, consider the quadratic form of matrix A given a nonzero vector $z \in \mathbb{R}^k$:

$$z^T A z = \left(\sum_{(u,v) \in E_A} W(u,v)(z(u) - z(v))^2 \right) + \left(\sum_{u \in V_A} z(u)^2 \sum_{(u,t) \in E, t \in V_C} W(u,t) \right).$$

The first term in parentheses is greater than or equal to 0 because the edge weights are positive. The second term in parentheses is strictly greater than 0. This occurs because (1) z is nonzero and (2) the graph G is connected; therefore, there must exist at least one edge (which must have a positive weight) between a vertex in V_A and a vertex in V_C. Combining these two results yields $z^T A z > 0$, i.e., A is positive definite.

We showed that matrix A is both symmetric and positive definite. This completes the proof because A is a principal submatrix of *any* reordered version of L (since permutation matrix X was arbitrary). □

This lemma shows that the AMLS algorithm can be used to decompose the graph Laplacian. Although our usage of the graph Laplacian is for reinforcement learning purposes, we emphasize the graph Laplacian is also useful more generally for dimensionality reduction [5] and clustering [76]. Thus, Lemma 3 shows that AMLS can be useful in those contexts as well for scaling to large problems.

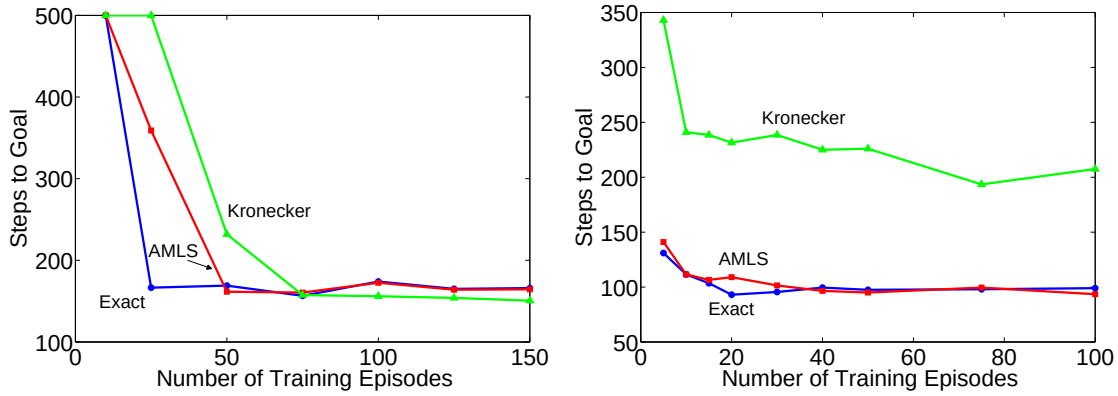
5.3.3 Experiments and Analysis

We repeated the mountain car and acrobot experiments described in Section 5.2.5 using basis functions computed by AMLS. The results demonstrate that learned policies perform similarly whether using "exact" eigenvectors or the AMLS approximate eigenvectors. This is an improvement over the basis functions derived from the Kronecker product approximation.

We examine the difference between eigenvectors and eigenvalues computed exactly, computed using the Kronecker product method, and computed using AMLS. The comparison demonstrates (1) the accuracy of the AMLS algorithm and (2) how imposing Kronecker structure can alter the space spanned by the eigenvectors. This analysis sheds some light on why the AMLS approximate eigenvectors performed well in mountain car and acrobot.

The experiments in mountain car and acrobot from Section 5.2.5 were repeated using the AMLS approximate eigenvectors as basis functions. We used the same parameters for graph construction (k, σ, weights) as detailed in Table 5.1. Likewise, we used the same number of eigenvectors for the AMLS method as we did when computing eigenvectors exactly (20 eigenvectors for mountain car and 25 eigenvectors for acrobot).

Thirty trials were run. Plots of the median performance of the learned policies are shown in Figure 5.7. For reference, the plots also include the results from Figure 5.4 for the "exact" basis functions and the "Kronecker" basis functions. The performance of policies learned using the AMLS basis functions is nearly the same as the performance of policies learned using the exact basis functions.



(a) Mountain Car

(b) Acrobot

Figure 5.7. Median performance over the 30 runs using the RPI algorithm. The basis functions are either derived from matrix A (Exact), from matrices B_R and C_R (Kronecker), or from the AMLS algorithm.

We examine the eigenvectors and eigenvalues computed exactly, using AMLS, and using the Kronecker method to help understand the performance differences. Since mountain car is a two dimensional continuous domain, it is easy to visually compare the eigenvectors.

Figure 5.8 shows the second through sixth eigenvectors for all three methods. The graph, which is from one of the 30 trials in the policy iteration experiments, contains 1050 vertices. The exact eigenvectors and those computed using AMLS are nearly identical. Notice there are some similarities (2nd, 5th, and 6th) and some differences (3rd and 4th) for the approximate eigenvectors computed using the Kronecker method.

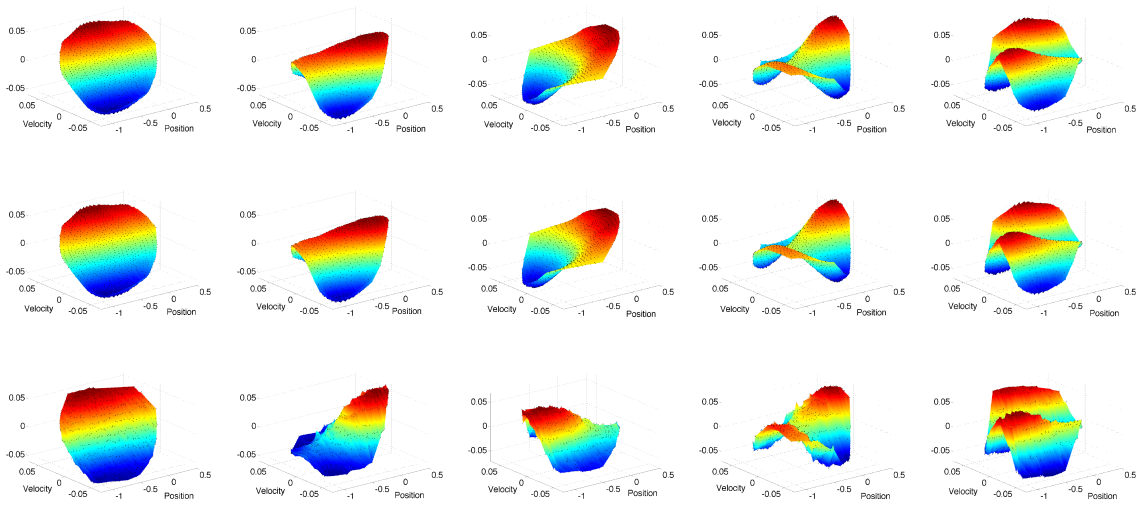


Figure 5.8. The 2nd-6th eigenvectors computed exactly (top row), computed using AMLS (middle row), and computed using the Kronecker method (bottom row) for the mountain car domain. The approximate eigenvectors computed using AMLS are nearly identical to the exact values.

The AMLS algorithm accurately computed 50 eigenvalues. The exact eigenvalues as well as those computed using AMLS are shown in Figure 5.9. The plot on the left of Figure 5.9 show the eigenvalues in increasing order. Notice the two curves are nearly identical. To detect small differences, the plot on the right of Figure 5.9 shows the difference between the eigenvalues, $(\lambda_i^{\text{AMLS}} - \lambda_i)$. This plot shows there is some discrepancy between the values and that the discrepancy is greater for larger eigenvalues. This behavior is to be expected

from AMLS because the projection method (i.e. using the eigenvectors from subproblems as a basis for approximating the larger eigenvalue problem) naturally captures more of the low frequency components of the Laplacian spectrum.

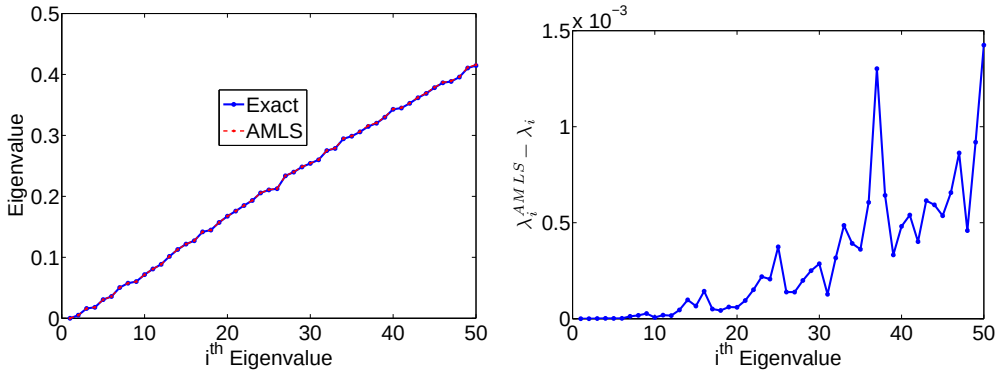


Figure 5.9. The first 50 eigenvalues of the normalized graph Laplacian for the mountain car task computed exactly and approximated using the AMLS algorithm (left). The difference between the approximate and exact eigenvalues (right) shows there is some discrepancy, but the error in the approximation is small relative to the absolute value.

Another way to compare the exact eigenvectors with the AMLS approximation is to compute the angle between the subspaces spanned by the eigenvectors. Before we define the angle between two subspaces, note it is easy to compute the angle between two vectors. Given two vectors x and y of the same length, one can compute the angle between x and y as $\arccos\left(\frac{x^T y}{\|x\|\|y\|}\right)$. Now, assume we have two subspaces S_X and S_Y. Using the definition of Bjorck and Golub [14], the angle between S_X and S_Y is defined to be the maximum angle between any vector in S_X and its closest vector in S_Y. This angle can be computed given orthonormal matrices X and Y spanning the subspaces S_X and S_Y respectively as:

$$\theta(S_X, S_Y) = \max_i \min_j \arccos\left(X_i^T Y_j\right)$$

with column indices i and j. Using this definition, the angle between the spaces spanned by the first 50 eigenvectors computed exactly and computed using AMLS is $\theta = 0.128$ radians,

or $\theta = 7.3^\circ$. Thus, the exact eigenvectors and the AMLS approximate eigenvectors span similar spaces.

The bottom row of plots in Figure 5.8 shows the second through sixth approximate eigenvectors produced using the Kronecker product method. Recall these eigenvectors are stored in a compressed form. They are formed by computing the Kronecker product of an eigenvector associated with a 105×105 matrix with an eigenvector of a 10×10 matrix. As described in Section 5.2.2, one can interpret the Kronecker product as partitioning the 1050 samples into 105 clusters, each of size 10. Practically, this means the eigenvectors produced using the Kronecker product method are more coarse ("blockier") than those produced using AMLS. This is evident in the jaggedness of the functions.

We also computed the angle between the subspaces spanned by the first 20 exact eigenvectors and the 20 eigenvectors computed using the Kronecker method. The angle was $\theta = 0.751$ radians, or $\theta = 43.0^\circ$. As expected, this is larger than the angle between the subspaces spanned by the exact and AMLS eigenvectors. However, $\theta = 43.0^\circ$ indicates there is still a significant degree of overlap between the exact and Kronecker product eigenspaces for mountain car. For graphs in acrobot, this angle was closer to 90° (meaning there was at least one function in the span of the true eigenvectors nearly orthogonal to all functions in the span of the approximate eigenvectors computed using the Kronecker method). This does not fully explain why the Kronecker product eigenvectors performed poorly as basis functions for the acrobot experiments, but it does provide evidence that graphs in acrobot may be poorly approximated with the block structure of the Kronecker product.

We also attempted to use the theoretical analysis developed in Section 5.2.4 to understand the behavior of the Kronecker product method. However, we found that for both mountain car and acrobot domains, the error in the Kronecker product approximation ($\|E\| = \|A - B \otimes C\|$) was greater than the eigengap of matrix A (d in Theorem 2). This violates one of the assumptions in the proof of Theorem 2.

In this section, we visually compared the eigenvectors produced by the three methods for the mountain car task, compared the eigenvalues produced using the exact and AMLS methods, and computed the angles between subspaces spanned by the eigenvectors. This analysis indicates the AMLS algorithm allows for a better approximation of the Laplacian eigendecomposition. The fact that AMLS allows for computing thousands of eigenvectors of sparse matrices with millions of entries makes it particularly attractive for generating proto-value functions. The analysis also shows the quality of the Kronecker product approximation depends more heavily on the specific graph being factorized. The Kronecker method's block structure allowed for a better approximation in the mountain car domain than in acrobot. Whether or not the Kronecker method's scalability and compression can be leveraged appears domain dependent.

5.4 Conclusions

In this chapter, we presented three ways to scale the graph-based basis construction method to larger problems. The greedy sampling procedure ensures the graph is constructed only using enough data points as necessary to ensure state space coverage. This allows for removing redundant samples. Not only does this accelerate graph and basis construction, but it also speeds up nearest neighbor searches when the features of a new state (not in the graph) are needed.

We also proposed two approximation algorithms for scaling up graph-based basis construction: the Kronecker product method and the Automated Multilevel Substructuring (AMLS) algorithm. Both methods can be used to compute approximate eigenvectors and the Kronecker product method can also be used to compute approximate diffusion wavelets. The Kronecker method decomposes the problem into smaller problems that are combined via the tensor product to approximate the original problem. Eigendecomposition or diffusion wavelet tree construction occurs on the smaller problems. This method has two substantial benefits: (1) basis construction only occurs on smaller matrices, and (2) the ap-

proximate eigenvectors or diffusion scaling and wavelet functions of the original problem are never explicitly formed/stored. To achieve these benefits, the method makes the strong assumption that the original problem has, to some degree, the block structure of the Kronecker product. The AMLS algorithm method does not make this assumption. Indeed, it can be used on any Laplacian matrix. AMLS recursively computes eigenvectors on smaller problems and then uses those solutions to approximate eigenvectors on larger problems. Experiments in the mountain car and acrobot tasks showed that the basis functions produced using AMLS resulted in very similar performance to the basis functions produced using exact eigendecomposition methods. We showed this was due to AMLS's accuracy in computing eigenvalues and eigenvectors. On the other hand, the results were mixed for the Kronecker product method. The basis functions allowed for policies that could reach the goal, albeit with some loss in performance, for mountain car. For the acrobot domain, the policies using Kronecker basis functions were significantly worse. This leads to the conclusion that the block structure of the Kronecker product allows for compression, but whether the compressed functions adequately represent the original problem appears task dependent.

The AMLS algorithm has been used on matrices with millions of rows to compute thousands of approximate eigenvectors. For some RL problems, a graph with millions of vertices could provide adequate coverage over the domain's state space. In these situations, we believe AMLS can be used to provide a useful set of basis functions for representing value functions. While AMLS can be used generally on all problems, the Kronecker product method is applicable to domains where some block structure exists and can be exploited.

CHAPTER 6

BASIS SELECTION

RL feature construction algorithms can be categorized into two types: one that iteratively generates basis functions based upon the current Bellman error, $T^{\pi}(\hat{V}) - \hat{V}$, and the other that generates a dictionary of basis functions.[1] Note the latter type requires a *selection* strategy to determine which elements from the dictionary to utilize. The graph-based methods studied in this dissertation are an example of the dictionary approach to basis construction. We propose three arguments for preferring this approach. First, a dictionary offers the flexibility of approximating value functions associated with many different policies. The other basis construction type iteratively generates basis functions for fitting just a single function based on the agent's current policy. Second, there is significant interest in the machine learning community on methods for generating *data-dependent dictionaries* [77, 60, 22, 59, 42]. By creating algorithms that operate on such dictionaries, we can naturally leverage future advances. Third, from a practical standpoint, we believe agents should construct representations that are general and useful in the pursuit of a variety of tasks. Over the course of an agent's lifetime, it should be able to reuse representations and knowledge from previous experience. The dictionary approach to basis construction is more in line with this ideal.

The previous sections of this dissertation have used a simple method for selecting which proto-value functions and diffusion wavelet functions to use when approximating a value function. The heuristic is to always use the K most global, or smoothest, basis functions. All prior applications of PVFs and diffusion wavelets in the literature have also used this

[1] We focus here on techniques for *explicitly* constructing features.

heuristic [67, 63, 68, 79, 44, 95].[2] This mechanism is independent of the policy being evaluated, meaning that all value functions are represented with the same set of basis functions. Using just the smoothest basis functions has the advantages of being computationally simple and robust to overfitting (although too much regularization can be just as problematic as too little regularization), but it does not exploit the full power of the basis function dictionary. In this chapter, we explore different selection mechanisms to better utilize the dictionary. This is an improvement over previous work for two reasons. First, it tailors the representation to the specific function being approximated. Second, tailoring the representation allows for using as few dictionary elements as possible, which is important for computational efficiency.

We evaluate four sparse basis selection algorithms: orthogonal matching pursuit (OMP) [82], order recursive matching pursuit (ORMP) [75], the LASSO [103], and least angle regression (LARS) [33]. Although we tested the selection algorithms using graph-based basis functions as a dictionary, the algorithms can be used with *any* set of basis functions. Each algorithm returns a subset of basis functions from the dictionary and a scalar coefficient associated with each selected basis function. The selected basis functions and coefficients are linearly combined to produce an approximate value function. We tested two different schemes for combining approximate policy evaluation and basis selection. The factor distinguishing these two schemes is whether the basis selection algorithm *directly* or *indirectly* uses the Bellman equation. These two schemes differ in terms of sparsity (how many basis functions are used in the approximate value function) and computational efficiency. To

[2] Mahadevan and Maggioni [66, 63] demonstrated the potential benefits for basis selection in a restricted setting where (1) the exact value function V^π is known, and (2) the dictionary is orthonormal (which is always the case for PVFs, but not for diffusion wavelets). In that setting, the best K basis functions are selected by finding the elements ϕ_i with the largest K values of $|\langle V^\pi, \phi_i \rangle|$. This results in the best rank-K approximation of V^π representable with the given dictionary. The result does not hold, however, when the dictionary is *not* orthogonal. We develop algorithms in this chapter that apply basis selection for arbitrary dictionaries and when the exact value function is unknown.

assess the combination of basis selection and approximate policy evaluation, both policy evaluation and policy iteration experiments were conducted.

6.1 Relevant Work

We provide a brief introduction to the basis selection problem and a few of the major algorithms. The basic formulation is that there is a signal $y \in \mathbb{R}^N$ to be represented with elements from an overcomplete dictionary $\Phi \in \mathbb{R}^{N \times K}$. Each basis function $\Phi_j \in \mathbb{R}^N$ has unit norm. The problem is to find a vector w such that $\Phi w = y$.[3] The decomposition of y is not unique; therefore, additional constraints are added to prefer solutions with certain qualities (e.g. sparseness, independence).

Two popular approaches to the sparse regression problem are matching pursuit and basis pursuit. Matching pursuit is an iterative, greedy algorithm whereas basis pursuit is an optimization principle that can be solved using any appropriate algorithm. Therefore, matching pursuit and basis pursuit are not mutually exclusive approaches to sparse regression.

6.1.1 Matching Pursuit

Matching pursuit (MP) [69] is a greedy algorithm that selects elements sequentially to best capture the signal. The algorithm begins with a coefficient vector w equal to all zeros and a residual vector y_{res} equal to the signal y. The first element is selected by scanning the dictionary and finding the largest correlation with the residual: $j^* \leftarrow \text{argmax}_j |\Phi_j^T y_{res}|, j \in [1, K]$. The coefficient for the selected basis function is adjusted: $w_{j^*} \leftarrow w_{j^*} + \Phi_{j^*}^T y_{res}$. Then the residual signal is computed $y_{res} \leftarrow y_{res} - (\Phi_{j^*}^T y_{res})\Phi_{j^*}$ and the process iterates. With MP, a basis function can be selected many times. There are other variants of MP, two of which are orthogonal matching pursuit (OMP) [82] and order recursive matching pursuit (ORMP) [75]. OMP differs from MP in the way the residual signal is computed.

[3] The model could also include a noise term, $\Phi w + e = y$.

OMP makes the residual orthogonal to the selected dictionary elements, which means OMP will never select the same dictionary element more than once whereas MP can. ORMP goes even further than OMP and adds the orthogonalization step into the selection process. Moghaddam et al. [73] proposed an efficient implementation of ORMP using partitioned matrix inverse techniques [39] and showed that sparse least-squares regression is equivalent to a generalized eigenvalue problem.

Algorithm 5 is a side-by-side comparison of the pseudocode for MP, OMP, and ORMP. We use the symbol $\mathcal{I}$ to refer to a set of indices in $[1, K]$ that indicate the elements of the dictionary Φ that are selected by the algorithm. Similarly, $w_{\mathcal{I}}$ refers to the scalar coefficients applied to the selected basis functions. Basis functions that are not selected have a scalar coefficient of 0. Thus, the signal y is approximated as $\Phi(:, \mathcal{I})w(\mathcal{I}) = \Phi_{\mathcal{I}} w_{\mathcal{I}}$.

Algorithm 5: `Variants of Matching Pursuit`

Input: Φ, y

Output: $\mathcal{I}$, $w_{\mathcal{I}}$ such that $\hat{y} \leftarrow \Phi_{\mathcal{I}} w_{\mathcal{I}}$

$\mathcal{I} \leftarrow \emptyset$, , $w \leftarrow \mathbf{0}$, $y_{res} \leftarrow y$

while (not done) **do**

If (matching pursuit)

$j^* \leftarrow \text{argmax}_j |\Phi_j^T y_{res}|$

$w_{j^*} \leftarrow w_{j^*} + \Phi_{j^*}^T y_{res}$

If $(w_{j^*} \neq 0)$, $\mathcal{I} \leftarrow \mathcal{I} \cup \{j^*\}$. **Else**, $\mathcal{I} \leftarrow \mathcal{I} - \{j^*\}$

$y_{res} \leftarrow y_{res} - (\Phi_{j^*}^T y_{res})\Phi_{j^*}$

If (orthogonal matching pursuit)

$j^* \leftarrow \text{argmax}_{j \notin \mathcal{I}} |\Phi_j^T y_{res}|$

$\mathcal{I} \leftarrow \mathcal{I} \cup \{j^*\}$

$w_{\mathcal{I}} \leftarrow (\Phi_{\mathcal{I}}^T \Phi_{\mathcal{I}})^{-1} \Phi_{\mathcal{I}}^T y$

$y_{res} \leftarrow y - \Phi_{\mathcal{I}} w_{\mathcal{I}}$

If (order recursive matching pursuit)

$j^* \leftarrow \text{argmin}_{j \notin \mathcal{I}} \|\Phi_{\mathcal{I}_{+j}} (\Phi_{\mathcal{I}_{+j}}^T \Phi_{\mathcal{I}_{+j}})^{-1} \Phi_{\mathcal{I}_{+j}}^T y - y\|^2$ where: $\mathcal{I}_{+j} \leftarrow \mathcal{I} \cup \{j\}$

$\mathcal{I} \leftarrow \mathcal{I} \cup \{j^*\}$

$w_{\mathcal{I}} \leftarrow (\Phi_{\mathcal{I}}^T \Phi_{\mathcal{I}})^{-1} \Phi_{\mathcal{I}}^T y$

end while

6.1.2 Basis Pursuit

Matching pursuit finds a sparse representation by greedily selecting the most promising elements. In contrast, basis pursuit (BP) [19] achieves sparsity by finding solutions to the following optimization problem: $\min \|w\|_1$ such that $\Phi w = y$. Sparsity of the solution comes from the use of L_1 norm. The BP problem can be solved using linear programming. Note the hard constraint $\Phi w = y$ is appropriate when the signal is noiseless. When the signal is noisy, it is appropriate to require $\|\Phi w - y\|^2$ to be small. The LASSO (least absolute shrinkage and selection operator) [103] implements this noisy version of basis pursuit in the following optimization problem: $\min \|y - \Phi w\|^2$ subject to $\|w\|_1 \leq k$. The LASSO can be solved using quadratic programming; however, a more efficient solution is to use the recently introduced least angle regression (LARS) algorithm [33] with a minor modification. LARS selects elements from the dictionary one at a time, much in the same way the matching pursuit algorithms work. The first element selected is the one that is most correlated with the signal y. Then LARS adjusts the weight on the first element until another element has as much correlation with the current residual. At that point, LARS includes this second element and then proceeds in a direction (i.e. changing the weights) *equiangular* between the first two elements. This strategy is less greedy than other algorithms that sequentially add dictionary elements. Interestingly, a small modification to the LARS algorithm produces the LASSO solution. While LARS by itself only adds basis functions at each step, this modification for LASSO gives the algorithm the ability to remove basis functions from the selected subset as well.

We evaluated the OMP, ORMP, LASSO, and LARS algorithms. It is easy to control the sparsity of each of these algorithms by limiting the number of basis functions that can be selected.

6.2 Combining Basis Selection and Approximate Policy Evaluation

The basis selection problem involves choosing elements from a dictionary to efficiently represent a target signal. The approximate policy evaluation problem is to represent the true value function V^π with an accurate approximation $\hat{V}$. If V^π were known, then basis selection could simply be performed with the target signal being V^π. However, V^π only becomes known through the Bellman equation: $V^\pi = R^\pi + \gamma P^\pi V^\pi = T^\pi(V^\pi)$. Thus, some framework is needed that effectively combines approximate policy evaluation (i.e. finding an accurate approximation $\hat{V}$) and basis selection (i.e. efficiently representing $\hat{V}$). We evaluate two schemes that achieve this combination. The difference between the two is in how they use the Bellman equation.[4] The first scheme uses the Bellman equation within the basis selection algorithm. This means that when the basis selection algorithm adjusts the weight vector w, this not only changes the approximation Φw *but also* changes the target signal based on a function of the Bellman equation. We call this the direct scheme because the selection algorithm directly encodes the Bellman equation. The second, or indirect, scheme does *not* use the Bellman equation within the basis selection algorithm. Rather, there is an iterative process that alternates between (1) setting the target signal using the Bellman equation, and (2) representing the target signal using the basis selection algorithm. These two schemes are described below in a very general form where:

1. $f\left(T^\pi(\Phi w') - \Phi w'\right)$ is a function f of the Bellman residual,
2. `BasisSelection` is an algorithm that selects dictionary elements $\mathcal{I}$ and computes weights $w_\mathcal{I}$ to minimize either $\left(f\left(T^\pi(\Phi w') - \Phi w'\right)\right)$ or $(y - \Phi w')$, and
3. `SetWeights` is an optional function that uses the dictionary elements determined by `BasisSelection`, but computes its own set of weights $w_\mathcal{I}$.

[4] Note the distinction we draw between the *direct* and *indirect* schemes is not new to RL. For example, the fitted Q-iteration algorithm [36] is an example of the indirect scheme, whereas the LARS-TD algorithm [52] is an example of the direct scheme. We are not aware of any other work that makes this distinction, so we introduced the terminology ourselves. Our analysis and experiments show that the direct and indirect schemes can behave very differently.

Direct Scheme	
$[\mathcal{I},\, w_{\mathcal{I}}] \leftarrow \texttt{BasisSelection}_{w'}\left(f\left(T^{\pi}(\Phi w') - \Phi w'\right)\right)$	
$w_{\mathcal{I}} \leftarrow \texttt{SetWeights}_{w'}\left(f\left(T^{\pi}(\Phi_{\mathcal{I}} w') - \Phi_{\mathcal{I}} w'\right)\right)$	OPTIONAL
$\hat{V} \leftarrow \Phi_{\mathcal{I}} w_{\mathcal{I}}$	

Indirect Scheme	
$\mathcal{I} \leftarrow \emptyset, \quad w_{\mathcal{I}} \leftarrow \emptyset$	
while (not converged)	
target $y \leftarrow T^{\pi}(\Phi_{\mathcal{I}} w_{\mathcal{I}})$	
$[\mathcal{I},\, w_{\mathcal{I}}] \leftarrow \texttt{BasisSelection}_{w'}\left(y - \Phi w'\right)$	
$w_{\mathcal{I}} \leftarrow \texttt{SetWeights}_{w'}\left(f\left(T^{\pi}(\Phi_{\mathcal{I}} w') - \Phi_{\mathcal{I}} w'\right)\right)$	OPTIONAL
$\hat{V} \leftarrow \Phi_{\mathcal{I}} w_{\mathcal{I}}$	

The direct and indirect schemes differ in their computational complexity and degree of sparsity. The computational complexity of the indirect scheme has the potential to be greater than the direct scheme because it iteratively calls the basis selection algorithm. This could be wasteful when the target signal given to the basis selection algorithm does not change significantly between iterations. On the other hand, the direct scheme, by using the Bellman residual as the target function for the basis selection algorithm, forces the regression algorithm to follow a specific path. To see this, consider the beginning of the basis selection algorithm when no basis functions have yet been selected. The Bellman residual is equal to the immediate reward function R^{π}. This means the first basis function selected is attempting to fit the immediate reward. For the sake of argument, assume the first basis function exactly fits the immediate reward. Now the Bellman residual is equal to the Bellman backup of the immediate reward, or $(T^{\pi}(R^{\pi}) - R^{\pi}) = \gamma P^{\pi} R^{\pi}$. This same logic can be used inductively to show basis selection proceeds in order of the elements in the Neumann series, $\sum_{i=0}^{\infty} (\gamma P^{\pi})^i R^{\pi}$.[5] Attempting to fit the elements in the Neumann series can lead to inefficient use of the basis functions. This occurs when there is structure

[5]For a bounded operator T, the Neumann series is defined as $\sum_{i=0}^{\infty} T^i$. One can show $\sum_{i=0}^{\infty} T^i = (I - T)^{-1}$. The value function V^{π} can be defined using the Neumann series as $V^{\pi} = (I - \gamma P^{\pi})^{-1} R^{\pi} = \sum_{i=0}^{\infty} (\gamma P^{\pi})^i R^{\pi}$.

in V^π that does not exist in the Neumann series; hence, the basis selection algorithm is unable to exploit the structure. Since the indirect scheme is not confined to this path, it has the potential to use fewer basis functions when representing the eventual approximate value function $\hat{V}$.

As an example of the potential inefficiency of the direct scheme, consider an undiscounted, deterministic chain MDP with an absorbing state at one end of the chain. Assume the reward function is 0 everywhere except $+1$ at the absorbing state. The optimal value function is a constant function equaling 1 in each state, but the Neumann series is a sequence of delta functions from one end of the chain to the other. Given a dictionary consisting of all the delta functions and a constant function, a basis selection algorithm implementing the direct scheme will select all the delta functions rather than the constant function. This may be an extreme example, but it is not uncommon for a MDP to have a spiky reward function that would cause similar behavior. Note this behavior can be particularly problematic for the multiscale diffusion wavelet dictionary where very localized basis functions (that are not necessary for representing V^π) can get selected before larger scale basis functions.

6.2.1 Direct Scheme

The next three sections outline the OMP-H_2 algorithm (i.e. OMP for basis selection and H_2 for setting the coefficients), the ORMP-H_2 algorithm, and the LASSO-H_2 and LARS-H_2 algorithms. Laplacian-based regularization is used in each algorithm. The LASSO-H_2 and LARS-H_2 algorithms are nearly identical, so we describe them simultaneously. Recall the BR and FP least-squares methods are easily instantiated by setting the hybrid parameter to $\xi = 1$ and $\xi = 0$ respectively.

Each algorithm takes as input a set of MDP samples $\{s_i, r_i, s'_i\}_{i=1}^n$, the discount factor γ, the hybrid parameter ξ, the dictionary Φ of basis functions, the graph Laplacian L along with its regularization parameter β_r, a distribution ρ over the states for weighting the least-

squares problem, and a maximum allowable number of basis functions k' that the algorithm can select. Each algorithm returns a set of indices $\mathcal{I}$ into the columns of Φ and scalar coefficients $w_{\mathcal{I}}$ such that the approximate value function $\hat{V} = \Phi_{\mathcal{I}} w_{\mathcal{I}}$. The sparsity of the solution is directly controlled by limiting the basis selection algorithm to at most $|\mathcal{I}| \leq k'$ basis functions. The parameter k' also limits the basis selection algorithm's computation and memory usage. Since the selection algorithm builds up sample-based estimates of the least-squares data structures (e.g. $\hat{A}^{-1}_{H_2,LR}$ and $\hat{b}_{H_2,LR}$), the size of the data structures cannot be larger than k'. This can be very important when the number of basis functions in the dictionary is large. To further speed up OMP-H_2, ORMP-H_2, LASSO-H_2, and LARS-H_2, we take advantage of the fact that the algorithms insert or remove one basis function at a time to the active set $\mathcal{I}$. The matrix $\hat{A}^{-1}_{\mathcal{I},\mathcal{I}}$ can be incrementally formed. However, to keep the pseudocode simple, the algorithms are not shown with this improvement. Appendix B describes how the algorithms can incrementally update $\hat{A}^{-1}_{\mathcal{I},\mathcal{I}}$. Note that within this chapter we only show pseudocode for the OMP-H_2 algorithm. The other algorithms are similarly described in Appendix C.

The OMP-H_2 and ORMP-H_2 algorithms terminate when either k' basis functions have been selected or when the change in the norm of the Bellman residual goes beneath a threshold.[6] The LASSO-H_2 and LARS-H_2 algorithms use both of those termination conditions as well as one other condition (related to the parameter k') that we discuss in that section.

6.2.1.1 Direct Scheme with Hybrid Method H_2

Algorithm 6 (OMP-H_2) shows the direct approach for combining orthogonal matching pursuit and the H_2 least-squares algorithm with Laplacian-based regularization. The algorithm maintains a sample-based estimate of the vector c where

[6] Using the terminology described in the algorithm boxes, the squared norm of the Bellman residual is written $\sum_{i=1}^{n} \rho(s_i) \left[r_i - (\phi_{\mathcal{I}}(s_i) + \beta_r \, g_{\mathcal{I}}(s_i) - \gamma \phi_{\mathcal{I}}(s'_i))^T w_{\mathcal{I}} \right]^2$. The change in the norm of the Bellman residual can easily be computed when inserting or removing a new basis function from the active set $\mathcal{I}$.

$$c_j = \left[(\Phi - \xi\gamma P^\pi \Phi)^T D_\rho (R^\pi - (\Phi - \gamma P^\pi \Phi) w) - \beta_r \Phi^T L D_\rho L \Phi w\right]_j$$
$$= \left[(\Phi - \xi\gamma P^\pi \Phi)^T D_\rho (R^\pi - (\Phi_\mathcal{I} - \gamma P^\pi \Phi_\mathcal{I}) w_\mathcal{I}) - \beta_r \Phi^T L D_\rho L \Phi_\mathcal{I} w_\mathcal{I}\right]_j . \qquad (6.1)$$

Each iteration of OMP-H_2 selects a new basis function to add to the active set by finding $j \notin \mathcal{I}$ that maximizes $|c_j|$. Then the weights $w_\mathcal{I}$ are adjusted to make the residual orthogonal to $\Phi_\mathcal{I}$.

Algorithm 6: OMP-H_2 with Laplacian-based Regularization

Input: $\{s_i, r_i, s'_i\}_{i=1}^n$, samples generated using policy π
$\phi : S \to \mathbb{R}^K$, basis function
$\rho : S \to \mathbb{R}^+$, weighting over the states
$\xi \in [0, 1]$, hybrid parameter ($\xi = 0$ is FP, $\xi = 1$ is BR)
L, graph Laplacian defined over states $\{s_i\}_{i=1}^n$ (graph edges denoted with $\sim$)
$\gamma \in [0, 1]$, discount factor
$\beta_r \in \mathbb{R}^+$, Laplacian-based regularization parameter
$k' \leq K$, maximum allowable number of basis functions

Output: $\mathcal{I}$, set of selected basis functions (indices into ϕ)
$w_\mathcal{I}$, weight vector such that $\hat{V}(s) = \phi_\mathcal{I}(s)^T w_\mathcal{I}$

$c \leftarrow \sum_{i=1}^n \rho(s_i)\phi(s_i) r_i$
Initialize active set $\mathcal{I} \leftarrow \emptyset$

while ($|\mathcal{I}| < k'$) and (Bellman residual not converged) **do**

1. Find most correlated inactive element:
   $j^* \leftarrow \text{argmax}_{j \notin \mathcal{I}}(|c_j|)$
2. Adjust active set:
   $\mathcal{I} \leftarrow \mathcal{I} \cup \{j^*\}$
3. Compute $\hat{A}_{\mathcal{I},\mathcal{I}}$ and $\hat{b}_\mathcal{I}$:
   $\hat{A}_{\mathcal{I},\mathcal{I}} \leftarrow \sum_{i=1}^n \rho(s_i)[(\phi_\mathcal{I}(s_i) - \xi\gamma\phi_\mathcal{I}(s'_i))(\phi_\mathcal{I}(s_i) - \gamma\phi_\mathcal{I}(s'_i))^T + \ldots$
   $\beta_r g_\mathcal{I}(s_i) g_\mathcal{I}(s_i)^T]$
   $\hat{b}_\mathcal{I} \leftarrow \sum_{i=1}^n \rho(s_i)(\phi_\mathcal{I}(s_i) - \xi\gamma\phi(s'_i)) r_i$
   where: $g(s_i) \leftarrow L(s_i, s_i)\, \phi(s_i)$
   $g(s_i) \leftarrow g(s_i) + L(s_i, s_{nbr})\, \phi(s_{nbr}), \qquad \forall\{s_{nbr} | s_{nbr} \neq s \wedge s \sim s_{nbr}\}$
4. Compute least-squares weights:
   $w_\mathcal{I} \leftarrow \hat{A}_{\mathcal{I},\mathcal{I}}^{-1} \hat{b}_\mathcal{I}$
5. Compute updated correlations:
   $c \leftarrow \sum_{i=1}^n \rho(s_i)[(\phi(s_i) - \xi\gamma\phi(s'_i))\left(r_i - (\phi_\mathcal{I}(s_i) - \gamma\phi_\mathcal{I}(s'_i))^T w_\mathcal{I}\right) - \ldots$
   $\beta_r\, g(s_i) g_\mathcal{I}(s_i)^T w_\mathcal{I}]$

end while

The next algorithm we consider is ORMP-H_2. We present the direct approach for combining ORMP and the H_2 least-squares algorithm with Laplacian-based regularization. This

is done to be consistent with our presentations of OMP-H_2, LASSO-H_2, and LARS-H_2, which helps make the pseudocode more readable since the H_2 least-squares data structures are identical from one algorithm to the next. However, we will show that it is only valid to combine ORMP and the BR least-squares method ($\xi = 1$). The pseudocode for ORMP-H_2 is provided in Appendix C (Algorithm 8).

The ORMP algorithm works by considering the impact each inactive basis function has on the least-squares problem. We use the terminology $\mathcal{I}_{+j}$ to indicate the inclusion of basis function j in the active set (i.e. $\mathcal{I}_{+j} \leftarrow \mathcal{I} \cup \{j\}$). The first step of Algorithm 8 determines the best inactive basis function $j \notin \mathcal{I}$ that maximizes $\left(\hat{b}^T_{\mathcal{I}_{+j}} \hat{A}^{-1}_{\mathcal{I}_{+j},\mathcal{I}_{+j}} \hat{b}_{\mathcal{I}_{+j}}\right)$.

Moghaddam et al. [73] point out that it is actually faster to find the inactive basis function that maximizes $\left(\hat{b}^T_{\mathcal{I}_{+j}} \hat{A}^{-1}_{\mathcal{I}_{+j},\mathcal{I}_{+j}} \hat{b}_{\mathcal{I}_{+j}} - \hat{b}^T_{\mathcal{I}} \hat{A}^{-1}_{\mathcal{I},\mathcal{I}} \hat{b}_{\mathcal{I}}\right)$ because some of the intermediate computation cancels out. The intermediate terms cancel due to properties of the partitioned matrix inverse. Note that since the extra term $\left(\hat{b}^T_{\mathcal{I}} \hat{A}^{-1}_{\mathcal{I},\mathcal{I}} \hat{b}_{\mathcal{I}}\right)$ is independent of all inactive basis functions, it does not alter the result of the maximization problem. ORMP-H_2 then inserts the best basis function into the active set, updates $\hat{A}^{-1}_{\mathcal{I},\mathcal{I}}$ and $\hat{b}_{\mathcal{I}}$, and iterates.

The ORMP algorithm merits further attention. This algorithm is particularly interesting because it uses the least-squares method to determine which basis function to include in the active set. The best basis function is determined by: $\operatorname{argmax}_{j \notin \mathcal{I}} \left(b^T_{\mathcal{I}_{+j}} A^{-1}_{\mathcal{I}_{+j},\mathcal{I}_{+j}} b_{\mathcal{I}_{+j}}\right)$. In other words, ORMP considers the impact of each inactive basis function on the least-squares problem. When the BR least-squares algorithm is used, the best basis function is:

$$
\begin{aligned}
j^* &\leftarrow \underset{j \notin \mathcal{I}}{\operatorname{argmax}} \left((b^{BR}_{\mathcal{I}_{+j}})^T (A^{BR}_{\mathcal{I}_{+j},\mathcal{I}_{+j}})^{-1} b^{BR}_{\mathcal{I}_{+j}}\right) \\
&\leftarrow \underset{j \notin \mathcal{I}}{\operatorname{argmax}} \left((b^{BR}_{\mathcal{I}_{+j}})^T w^{BR}_{\mathcal{I}_{+j}}\right) \\
&\leftarrow \underset{j \notin \mathcal{I}}{\operatorname{argmax}} \left((R^\pi)^T D_\rho (\Phi_{\mathcal{I}_{+j}} - \gamma P^\pi \Phi_{\mathcal{I}_{+j}}) w^{BR}_{\mathcal{I}_{+j}}\right) \\
&\leftarrow \underset{j \notin \mathcal{I}}{\operatorname{argmax}} \langle R^\pi,\, \hat{V}^{BR}_{\mathcal{I}_{+j}} - \gamma P^\pi \hat{V}^{BR}_{\mathcal{I}_{+j}} \rangle_\rho
\end{aligned}
$$

where $\langle \cdot, \cdot \rangle_\rho$ denotes the ρ-weighted inner product. This makes intuitive sense since the BR least-squares problem is fitting a function $\hat{V}^{BR}$ that minimizes $\|R^\pi + \gamma P^\pi \hat{V}^{BR} - \hat{V}^{BR}\|_\rho^2$. Now consider the direct scheme for combining ORMP and the FP least-squares algorithm. One can show the best inactive basis function for ORMP-FP is: $\operatorname{argmax}_{j \notin \mathcal{I}} \langle R^\pi, \hat{V}^{FP}_{\mathcal{I}_{+j}} \rangle_\rho$. This maximization does not make sense since selecting basis functions using this criteria leads to a value function that approximates the reward R^π. A simple idea to try to rescue ORMP-FP is to change the maximization to: $\operatorname{argmax}_{j \notin \mathcal{I}} \left((b^{BR}_{\mathcal{I}_{+j}})^T (A^{FP}_{\mathcal{I}_{+j}, \mathcal{I}_{+j}})^{-1} b^{FP}_{\mathcal{I}_{+j}} \right)$. Notice the use of the two different vectors $b^{BR}_{\mathcal{I}_{+j}}$ and $b^{FP}_{\mathcal{I}_{+j}}$. This leads to selecting basis functions according to: $\operatorname{argmax}_{j \notin \mathcal{I}} \langle R^\pi, \hat{V}^{FP}_{\mathcal{I}_{+j}} - \gamma P^\pi \hat{V}^{FP}_{\mathcal{I}_{+j}} \rangle_\rho$. Although this is seemingly more valid than the original formulation, it is still problematic. The underlying problem is that the FP objective function $\|\Pi_\rho(R^\pi + \gamma P^\pi \hat{V}^{FP}) - \hat{V}^{FP}\|_\rho^2$ can always be set to 0 for any set of basis functions.

One must be careful when directly combining least-squares policy evaluation algorithms and basis selection algorithms. The result of this analysis is that ORMP-FP is **not** valid but ORMP-BR is valid. However, ORMP can be used with both FP and BR in the *indirect* scheme described in Section 6.2.2.

The last two direct algorithms that we consider are LASSO-H_2 and LARS-H_2. To achieve sparsity, the LASSO algorithm takes the loss function from Equation 4.4 and includes an L_1 constraint on the coefficient vector. This takes the form:

$$w_{H_2,LR} = \underset{w' \in \mathbb{R}^K}{\operatorname{argmin}} \Big(\frac{\xi}{2} \|T^\pi(\Phi w') - \Phi w'\|_\rho^2 + \frac{1-\xi}{2} \|T^\pi(u) - \Phi w'\|_\rho^2 + \ldots + \frac{\beta_r}{2} \|L \Phi w'\|_\rho^2 + \beta_s \|w'\|_1 \Big) \tag{6.2}$$

where $\beta_s \in \mathbf{R}^+$ is a regularization parameter that dictates the sparsity of the solution. Larger values of β_s result in a coefficient vector w with more zero entries. In fact, there exists a value of β_s for which the resulting vector w has all zero entries.

Loth et al. [62] and Kolter and Ng [52] recently proposed using the LASSO algorithm for approximate policy evaluation. Our description of the algorithm and its derivation follows along the same lines as that of Kolter and Ng [52]. The only exception is that we consider Laplacian-based regularization and they did not. Therefore, our LASSO-H_2 algorithm with $\xi = 0$ and $\beta_r = 0$ exactly coincides with their algorithm.[7]

The minimization problem in Equation 6.2 can be converted into the following set of optimality conditions:

$$\begin{aligned} -\beta_s \leq c_j \leq \beta_s \quad \forall j \\ c_j = \beta_s \Rightarrow w_j \geq 0 \\ c_j = -\beta_s \Rightarrow w_j \leq 0 \\ -\beta_s < c_j < \beta_s \Rightarrow w_j = 0, \end{aligned} \tag{6.3}$$

where variable c_j is defined according to Equation 6.1. The LASSO-H_2 algorithm continually adjusts the weight vector (by adding or subtracting basis functions from the active set) while satisfying the optimality conditions. The algorithm is initialized with $\mathcal{I} \leftarrow \emptyset$ and $w \leftarrow \mathbf{0}$. The optimality conditions can be satisfied with this initialization for some $\bar{\beta}_s > \beta_s$. The algorithm proceeds to reduce $\bar{\beta}_s$ (by inserting basis functions into $\mathcal{I}$ and adjusting $w_{\mathcal{I}}$) while satisfying the optimality conditions until $\bar{\beta}_s = \beta_s$ or some other termination criteria is triggered. The other termination criteria we used were a maximum number of basis functions (k') and a threshold on the change in the norm of the Bellman residual.

The optimality conditions ensure that $|c_{\mathcal{I}}| = \bar{\beta}_s$ for all basis functions in the active set. This property is maintained by changing the weight vector according to:

[7] Our terminology is slightly different from that used by Kolter and Ng [52]. Their LARS-TD algorithm is the same as our LASSO-H_2 algorithm with $\xi = 0$ and $\beta_r = 0$. The distinction we draw between LARS and LASSO is whether the algorithm only adds basis functions to the active set (LARS) or both adds and removes basis functions (LASSO).

$$\Delta w_{\mathcal{I}} = \left[(\Phi_{\mathcal{I}} - \xi\gamma P^{\pi}\Phi_{\mathcal{I}})^T D_{\rho}(\Phi_{\mathcal{I}} - \gamma P^{\pi}\Phi_{\mathcal{I}}) + \beta_r \Phi_{\mathcal{I}}^T L D_{\rho} L \Phi_{\mathcal{I}}\right]^{-1} \text{sign}(c_{\mathcal{I}}),$$

where sign$(c_{\mathcal{I}})$ replaces the entries in $c_{\mathcal{I}}$ with values ± 1 depending on the sign. The change in the weight vector $\Delta w_{\mathcal{I}}$ dictates how the vector c changes:

$$\Delta c = \left((\Phi - \xi\gamma P^{\pi}\Phi)^T D_{\rho}(\Phi_{\mathcal{I}} - \gamma P^{\pi}\Phi_{\mathcal{I}}) + \beta_r \Phi^T L D_{\rho} L \Phi_{\mathcal{I}}\right) \Delta w_{\mathcal{I}}.$$

The vector Δc allows one to compute if and when an inactive basis function $j \notin \mathcal{I}$ will have a value c_j that reaches the same value as those in the active set. The first inactive basis function that reaches this point is computed as:

$$[\alpha^*,\, j^*] = [\min{}^+,\, \text{argmin}]_{j \notin \mathcal{I}} \left(\frac{c_j - \bar{\beta}_s}{\Delta c_j - 1},\, \frac{c_j + \bar{\beta}_s}{\Delta c_j + 1} \right),$$

where $\min^+$ indicates the minimization is only over positive values, α^* is the minimizing value, and j^* is the minimizing argument.

Before adding basis function j^* to the active set, the LASSO-H$_2$ algorithm must check to see whether an element in the active set $j \in \mathcal{I}$ has a coefficient w_j differing in sign with c_j as such an event would violate the optimality conditions.[8] The first active basis function that reaches this point is computed as:

$$[\alpha^{\#},\, j^{\#}] = [\min{}^+,\, \text{argmin}]_{j \in \mathcal{I}} \left(-\frac{w_j}{\Delta w_j} \right).$$

If all elements in the minimization are negative, then $\alpha^{\#}$ is set to ∞. If the step size $\alpha^* < \alpha^{\#}$, then basis function j^* is added to the active set. If the reverse is true, then basis function $j^{\#}$ is removed from the active set. Pseudocode for LARS-H$_2$ and LASSO-H$_2$ is given in Appendix C (Algorithm 9).

[8] Note this is the only difference between LASSO-H$_2$ and LARS-H$_2$. LARS-H$_2$ is not required to ensure w_j and c_j have the same sign. Therefore, LARS-H$_2$ does not remove basis functions from the active set.

The LARS-H_2 and LASSO-H_2 algorithms adjust the coefficient vector $w_{\mathcal{I}}$ in an equiangular direction. This means that the residual is never made completely orthogonal with the selected basis functions $\Phi_{\mathcal{I}}$. A common "fix" to this issue is to enforce orthogonality once LARS-H_2 and LASSO-H_2 terminate. We list this as an optional step at the end of the algorithm.

6.2.1.2 Direct Scheme with Hybrid Method H_1

The previous three sections described the OMP-H_2, ORMP-H_2, LASSO-H_2, and LARS-H_2 algorithms. By setting the hybrid parameter ξ to 0 or 1, these algorithms implement the FP and BR objective functions. We describe here how the algorithms would change to handle the H_1 objective function. We do this in detail for OMP and then simply highlight where the (similar) changes need to be made in ORMP, LASSO, and LARS.

The memory and computation requirements are identical whether using the FP, BR, or H_2 least-squares criteria. The hybrid algorithm H_1 however requires more memory and computation time. As shown in the equations below, H_1 requires forming two matrices of size $K \times K$ where K is the number of basis functions in the dictionary. This can be prohibitively large depending on the size of the dictionary. Note that all basis selection algorithms when using FP, BR, and H_2 do not form matrices larger than $k' \times k'$ where $k' \leq K$ is specified by the user to be the maximum number of basis functions that the algorithm can select.

The following four lines of Algorithm 6 (OMP-H_2) would need to change to accommodate the H_1 objective function.

1. The first time c is initialized:

   $c \leftarrow \xi \hat{b}_{BR} + (1-\xi)(\hat{A}^{FP})^T \hat{C}^{-1} \hat{b}_{FP}$

   where: $\hat{b}_{BR} \leftarrow \sum_{i=1}^{n} \rho(s_i)(\phi(s_i) - \gamma\phi(s_i'))r_i$

   $\hat{b}_{FP} \leftarrow \sum_{i=1}^{n} \rho(s_i)\phi(s_i)r_i$

$$\hat{A}^{FP} \leftarrow \sum_{i=1}^{n} \rho(s_i) \left[\phi(s_i)(\phi(s_i) - \gamma\phi(s_i'))^T + \beta_r g(s_i) g(s_i)^T\right]$$
$$\hat{C} \leftarrow \sum_{i=1}^{n} \rho(s_i)\phi(s_i)\phi(s_i)^T \ .$$

2. Computing $\hat{A}_{\mathcal{I},\mathcal{I}}$ in Step 3:

$$\hat{A}_{\mathcal{I},\mathcal{I}} \leftarrow \xi \hat{A}^{BR}_{\mathcal{I},\mathcal{I}} + (1-\xi)(\hat{A}^{FP}_{\mathcal{I},\mathcal{I}})^T \hat{C}^{-1}_{\mathcal{I},\mathcal{I}} \hat{A}^{FP}_{\mathcal{I},\mathcal{I}}$$

where: $\hat{A}^{BR}_{\mathcal{I},\mathcal{I}} \leftarrow \sum_{i=1}^{n} \rho(s_i) \left[(\phi_{\mathcal{I}}(s_i) - \gamma\phi_{\mathcal{I}}(s_i'))(\phi_{\mathcal{I}}(s_i) - \gamma\phi_{\mathcal{I}}(s_i'))^T + \beta_r g_{\mathcal{I}}(s_i) g_{\mathcal{I}}(s_i)^T\right]$

$$\hat{A}^{FP}_{\mathcal{I},\mathcal{I}} \leftarrow \sum_{i=1}^{n} \rho(s_i) \left[\phi_{\mathcal{I}}(s_i)(\phi_{\mathcal{I}}(s_i) - \gamma\phi_{\mathcal{I}}(s_i'))^T + \beta_r g_{\mathcal{I}}(s_i) g_{\mathcal{I}}(s_i)^T\right]$$
$$\hat{C}_{\mathcal{I},\mathcal{I}} \leftarrow \sum_{i=1}^{n} \rho(s_i)\phi_{\mathcal{I}}(s_i)\phi_{\mathcal{I}}(s_i)^T \ .$$

3. Computing $\hat{b}_{\mathcal{I}}$ in Step 3:

$$\hat{b}_{\mathcal{I}} \leftarrow \xi \hat{b}^{BR}_{\mathcal{I}} + (1-\xi)(\hat{A}^{FP}_{\mathcal{I},\mathcal{I}})^T \hat{C}^{-1}_{\mathcal{I},\mathcal{I}} \hat{b}^{FP}_{\mathcal{I}}$$

where: $\hat{b}^{BR}_{\mathcal{I}} \leftarrow \sum_{i=1}^{n} \rho(s_i)(\phi_{\mathcal{I}}(s_i) - \gamma\phi_{\mathcal{I}}(s_i'))r_i$

$$\hat{b}^{FP}_{\mathcal{I}} \leftarrow \sum_{i=1}^{n} \rho(s_i)\phi_{\mathcal{I}}(s_i) r_i$$
$$\hat{A}^{FP}_{\mathcal{I},\mathcal{I}} \leftarrow \sum_{i=1}^{n} \rho(s_i) \left[\phi_{\mathcal{I}}(s_i)(\phi_{\mathcal{I}}(s_i) - \gamma\phi_{\mathcal{I}}(s_i'))^T + \beta_r g_{\mathcal{I}}(s_i) g_{\mathcal{I}}(s_i)^T\right]$$
$$\hat{C}_{\mathcal{I},\mathcal{I}} \leftarrow \sum_{i=1}^{n} \rho(s_i)\phi_{\mathcal{I}}(s_i)\phi_{\mathcal{I}}(s_i)^T \ .$$

4. Updating c in Step 5:

$$c \leftarrow \xi c_{BR} + (1-\xi)(\hat{A}^{FP})^T \hat{C}^{-1} c_{FP}$$

where: $c_{BR} \leftarrow \sum_{i=1}^{n} \rho(s_i)[(\phi(s_i) - \gamma\phi(s_i'))(r_i - (\phi_{\mathcal{I}}(s_i) - \gamma\phi_{\mathcal{I}}(s_i'))^T w_{\mathcal{I}}) - \ldots$

$$\beta_r g(s_i) g_{\mathcal{I}}(s_i)^T w_{\mathcal{I}}]$$
$$c_{FP} \leftarrow \sum_{i=1}^{n} \rho(s_i) \left[\phi(s_i)(r_i - (\phi_{\mathcal{I}}(s_i) - \gamma\phi_{\mathcal{I}}(s_i'))^T w_{\mathcal{I}}) - \beta_r g(s_i) g_{\mathcal{I}}(s_i)^T w_{\mathcal{I}}\right]$$
$$\hat{A}^{FP} \leftarrow \sum_{i=1}^{n} \rho(s_i) \left[\phi(s_i)(\phi(s_i) - \gamma\phi(s_i'))^T + \beta_r g(s_i) g(s_i)^T\right]$$
$$\hat{C} \leftarrow \sum_{i=1}^{n} \rho(s_i)\phi(s_i)\phi(s_i)^T \ .$$

The changes to ORMP, LARS, and LASSO are very similar to the changes made for OMP; therefore, we just point out the lines that need to be edited. For ORMP, four lines would need to change: computing $\hat{b}_{\mathcal{I}_{+j}}$ in Step 1, computing $\hat{A}_{\mathcal{I}_{+j},\mathcal{I}_{+j}}$ in Step 1, computing $\hat{A}_{\mathcal{I},\mathcal{I}}$ in Step 3, and computing $\hat{b}_{\mathcal{I}}$ in Step 3. For LARS and LASSO, four lines would need to change: the first time c is initialized, computing $\hat{A}_{\mathcal{I},\mathcal{I}}$ in Step 1, computing Δc in Step 2, and computing $\hat{b}_{\mathcal{I}}$ at the final optional step of the algorithm.

6.2.2 Indirect Scheme

The indirect scheme uses an iterative approach to sparse approximate policy evaluation. The iterative approach alternates between (1) setting the target function using the Bellman backup operator, and (2) representing the the target function using the basis selection algorithm. This potentially makes the indirect scheme more computationally intensive than the direct scheme, but it frees up the basis selection algorithm to choose the best basis functions for fitting the approximate value function (instead of fitting the ordered elements in the Neumann series). We describe the iterative, indirect scheme in Algorithm 7. This is a general framework which can utilize any sparse basis selection (regression) algorithm. The sparse basis selection algorithm is denoted as input ***BSel***(y) where y is the target function that ***BSel*** fits using dictionary Φ. For ***BSel***, we evaluated the pure regression versions of OMP, ORMP, LASSO, and LARS with the only exception being they were augmented to include Laplacian-based regularization. The pure regression versions of OMP and ORMP without regularization were described in Algorithm 5.

6.3 Action-Value Function Approximation

The previous two sections described the direct and indirect schemes for approximating the value function. The same algorithms can also be used to approximate the action-value function. The graph-based basis functions, which are defined just over states, can be also used to approximate the action-value function. This is accomplished by using the basis functions for each discrete action. For example, consider a MDP with two actions, a_1 and a_2. The approximate action-value function $\hat{Q}$ can take the form:

$$\hat{Q} = \begin{bmatrix} \hat{Q}(\cdot, a_1) \\ \hat{Q}(\cdot, a_2) \end{bmatrix} = \begin{bmatrix} \Phi_{\mathcal{I}_{a_1}} & \mathbf{0} \\ \mathbf{0} & \Phi_{\mathcal{I}_{a_2}} \end{bmatrix} \begin{bmatrix} w_{\mathcal{I}_{a_1}} \\ w_{\mathcal{I}_{a_2}} \end{bmatrix} = \Phi_{\mathcal{I}}\, w_{\mathcal{I}}.$$

Algorithm 7: `Indirect Scheme for Sparse Approx. Pol. Eval.`

Input: $\{s_i, r_i, s'_i\}_{i=1}^n$, samples generated using policy π
$\phi : S \to \mathbb{R}^K$, basis function
$\rho : S \to \mathbb{R}^+$, weighting over the states
L, graph Laplacian defined over states $\{s_i\}_{i=1}^n$ (graph edges denoted with $\sim$)
$\gamma \in [0, 1]$, discount factor
$\beta_r \in \mathbb{R}^+$, Laplacian-based regularization parameter
$maxIter \in \mathbb{N}$, maximum number of iterations
BSel(y), sparse basis selection algorithm that approximates a target function y using the dictionary ϕ. The termination criteria for ***BSel*** includes:
$k' \leq K$, maximum allowable number of basis functions
a threshold on the residual $\|y - \Phi w\|_\rho^2$
any other algorithm specific parameters (e.g. β_s for LASSO)

Output: $\mathcal{I}$, set of selected basis functions (indices into ϕ)
$w_\mathcal{I}$, weight vector such that $\hat{V}(s) = \phi_\mathcal{I}(s)^T w_\mathcal{I}$

Initialize active set $\mathcal{I} \leftarrow \emptyset$, $\hat{w}_\mathcal{I} \leftarrow \emptyset$, $iter \leftarrow 0$

while $(iter < maxIter)$ and (Bellman residual not converged) **do**

1. Form target vector y using the sampled Bellman backup:
   $y_i \leftarrow r_i + \gamma \phi_\mathcal{I}(s'_i)^T w_\mathcal{I} \qquad \forall i$
2. Run the sparse basis selection (regression) algorithm to fit y:
   $[\mathcal{I}, w_\mathcal{I}] \leftarrow$ ***BSel***(y)
3. OPTIONAL: Adjust $w_\mathcal{I}$ using one of the least-squares methods:
   $w_\mathcal{I} \leftarrow \hat{A}_{\mathcal{I},\mathcal{I}}^{-1} \hat{b}_\mathcal{I}$
   For example, if using FP least-squares method, then:
   $\hat{A}_{\mathcal{I},\mathcal{I}} \leftarrow \sum_{i=1}^n \rho(s_i) \left[\phi_\mathcal{I}(s_i)(\phi_\mathcal{I}(s_i) - \gamma \phi_\mathcal{I}(s'_i))^T + \beta_r g_\mathcal{I}(s_i) g_\mathcal{I}(s_i)^T\right]$
   $\hat{b}_\mathcal{I} \leftarrow \sum_{i=1}^n \rho(s_i) \phi_\mathcal{I}(s_i) r_i$
4. Increment the iteration count:
   $iter \leftarrow iter + 1$

end while

Notice the approximate action-value function can use a different set of basis functions for each action: $\hat{Q}(\cdot, a_1)$ uses the basis functions indexed by $\mathcal{I}_{a_1}$ and $\hat{Q}(\cdot, a_2)$ uses basis functions indexed by $\mathcal{I}_{a_2}$.

Algorithms 6, 7, 8, and 9 can be used with this definition without changing any steps. However, if these algorithms are used without changes, the number of selected basis functions per action may not be equal. For the MDP with two actions a_1 and a_2, this means $|\mathcal{I}_{a_1}|$ will not necessarily be equal to $|\mathcal{I}_{a_2}|$. It may be desirable to require the number of basis functions per action to be equal (or approximately equal). This constraint can easily be added to the indirect scheme (Algorithm 7) and to the direct schemes involving OMP

and ORMP (Algorithms 6 and 8). It does not seem possible to add this constraint to the direct scheme involving LASSO and LARS (Algorithm 9) because of the way these algorithms control the correlation between the basis functions and the target function. For example, step 3 of Algorithm 9 relies on the fact that all basis functions *not* in $\mathcal{I}$ (i.e. basis functions that have not been selected) have a correlation $|c_j| < \bar{\beta}_s$. Adding a constraint that the number of basis functions per action should be roughly equal (which would entail changing step 3 to not just select the minimizing element) would seem to break this logic.

Algorithms 6, 7, 8, and 9 can produce approximate action-value functions for a specific policy. These algorithms can also be used within least-squares policy iteration (LSPI) [56]. One LSPI iteration takes a batch of MDP samples $\{s_i, a_i, r_i, s'_i\}_{i=1}^n$ and a policy π and produces $\hat{Q}$, an approximation of Q^π. The greedy policy implicitly defined by $\hat{Q}$ is then used in the next iteration of LSPI.

6.4 Experiments

6.4.1 Approximate Policy Evaluation

The following components were varied in the experiments:

- least-squares method (FP, BR, and H_2)
- basis selection method (OMP, ORMP, LASSO, and LARS)
- scheme for sparse approximate policy evaluation (direct and indirect)
- amount of Laplacian-based regularization (β_r)
- dictionary (PVFs and diffusion wavelet functions).

To get a solid understanding of how each component influences the policy evaluation problem, we chose the 50 state chain MDP [56]. This domain is easily visualized. The problem consists of 50 states ($s_i, i \in [1, 50]$) and two actions moving the agent left ($s_i \rightsquigarrow$

s_{i-1}) or right ($s_i \rightsquigarrow s_{i+1}$). The reward function is defined as $+1$ in states s_{10} and s_{41} and zero everywhere else. The discount factor is $\gamma = 0.9$.

We consider the task of evaluating the optimal policy π^*. Rather than sampling from π^* to generate a data set, we used the true model P^{π^*} and R^{π^*} in the following experiments. This choice was made to remove the influence of sampling so that we can adequately compare and contrast performance. However, we note that using the model rather than samples eliminates the bias of the BR method.

The graph used to form the PVFs and diffusion wavelets consists of 50 vertices with self-edges and edges between "adjacent" vertices. The PVF dictionary, which was constructed using the combinatorial Laplacian, consists of 50 global basis functions. The diffusion wavelet tree was constructed using the parameter $\epsilon = 10^{-4}$. The number of scaling and wavelet functions is shown in Table 6.4.1. We evaluated three dictionaries constructed

Tree Level j	$\lvert\psi_{j-1}\rvert$	$\lvert\phi_j\rvert$
1	0	50
2	9	41
3	13	28
4	7	21
5	5	16
6	5	11
7	3	8
8	2	6
9	2	4
10	1	3

Table 6.1. Number of wavelet and scaling functions at each tree level for the 50 state chain MDP.

from this tree. The first dictionary consisted of all 235 functions in the tree (47 wavelet and 188 scaling functions). The second dictionary consisted of the 135 functions at tree level 3 or greater (38 wavelet and 97 scaling functions). The 100 extra functions in the first dictionary consist of very localized basis functions as well as some oscillatory functions. Note that both the first and second dictionaries are overcomplete, so selecting elements from these dictionaries can lead to linear dependence in the basis functions. The third dictionary

consisted of all 47 wavelet functions and the 3 scaling functions at tree level 10. This third dictionary is orthonormal whereas the first two dictionaries are overcomplete. A further optimization that we did not pursue would be to select the "best" such orthonormal dictionary (amongst the 10 possible orthonormal dictionaries) instead of just using the dictionary that reaches to tree level 10.

We systematically tested different combinations of dictionary, least-squares algorithm, policy evaluation scheme, amount of Laplacian regularization, and basis selection method. The list of these combinations is shown in Table 6.4.1. We present the main findings of these experiments along with supporting figures. For a description of all the experiments and resulting value function plots, we refer the reader to our technical report [45].

The result of each experiment is an approximate value function $\hat{V}$. Rather than simply report a number (such as the Bellman residual norm, $\|T^{\pi}(\hat{V}) - \hat{V}\|$, or the true error, $\|V^{*} - \hat{V}\|$), we found it much more illuminating to qualitatively assess the approximate value functions. This leads to some interesting insights into the interaction among the basis selection algorithm, least-squares method, and dictionary. The policy iteration experiments in the next section provide a more quantitative measure of performance.

We summarize the policy evaluation experiments with the following findings.

- OMP-FP & the effect of Laplacian regularization

  Figure 6.1 shows the results of using the OMP-FP algorithm, a varying number of basis functions (4, 8, and 12), and a different amount of Laplacian regularization ($\beta_r = 0$ and $\beta_r = 0.1$). The captions under the plots show the different dictionaries used to produce the approximate value function. We use the shorthand DWT(50) to refer to the diffusion wavelet dictionary with 50 orthonormal bases, DWT(135) to refer to the diffusion wavelet dictionary with 135 functions at tree level 3 or greater, and DWT(235) to refer to the dictionary containing all 235 scaling and wavelet functions in the tree.

Scheme	Algorithm	Dictionary
Direct	OMP-FP, LASSO-FP	PVFs
Direct	OMP-BR, ORMP-BR, LASSO-BR	PVFs
Direct	OMP-H_2	PVFs
Indirect FP & BR	OMP	PVFs
Indirect FP & BR	ORMP	PVFs
Indirect FP & BR	LASSO	PVFs
Direct	OMP-FP, LASSO-FP, LARS-FP	235 Diffusion Wavelets
Direct	OMP-FP, LASSO-FP, LARS-FP	135 Diffusion Wavelets
Direct	OMP-FP, LASSO-FP	50 Orthog. Diffusion Wavelets
Direct	ORMP-BR	235 Diffusion Wavelets
Direct	OMP-BR, ORMP-BR, LASSO-BR	135 Diffusion Wavelets
Direct	OMP-BR, ORMP-BR, LASSO-BR	50 Orthog. Diffusion Wavelets
Indirect FP	OMP, ORMP, LASSO, LARS	235 Diffusion Wavelets
Indirect FP	OMP, ORMP, LASSO, LARS	135 Diffusion Wavelets
Indirect FP	OMP, ORMP, LASSO	50 Orthog. Diffusion Wavelets
Indirect BR	OMP, ORMP, LASSO, LARS	235 Diffusion Wavelets
Indirect BR	OMP, ORMP, LASSO, LARS	135 Diffusion Wavelets
Indirect BR	OMP, ORMP, LASSO	50 Orthog. Diffusion Wavelets

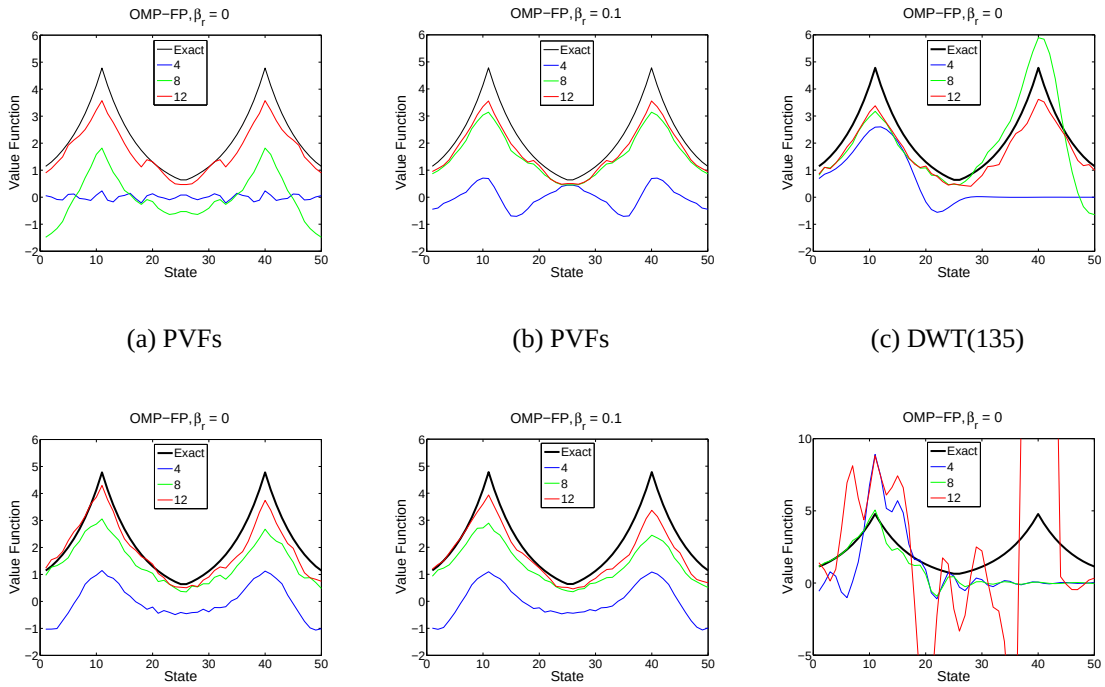
Table 6.2. Parameters varied in the policy evaluation experiments for the 50 state chain MDP.

The approximate value functions learned using either of the orthogonal dictionaries (PVFs or DWT(50)) accurately captured the shape of the exact value function V^* when using at least 8 basis functions. The approximations produced using the diffusion wavelet dictionary DWT(50) were more accurate than those using the PVF dictionary. In fact, even using just 4 basis functions from the DWT(50) dictionary resulted in an approximate value function that tracked the shape (not magnitude) of V^*.

The approximate value function learned using DWT(235) became unstable when 12 basis functions were used. This occurs because the matrix $\hat{A}_{\mathcal{I},\mathcal{I}}^{-1}$ became nearly singular. The results were slightly better when using the DWT(135) dictionary, which removes some of the most localized and oscillatory functions in DWT(235). This indicates the aggressiveness of OMP-FP may be a potential problem with a highly

overcomplete dictionary. It is possible though to make the algorithm more stable by checking the condition number of $\hat{A}^{-1}_{\mathcal{I},\mathcal{I}}$ before inserting a basis function.

Lastly, notice the influence Laplacian regularization has on the value functions produced from the PVF and DWT(50) dictionaries. The approximations with regularization ($\beta_r = 0.1$) clearly are smoother with respect to the topology of the chain. This had a noticeable effect on the basis selection process for the PVF dictionary.



(a) PVFs (b) PVFs (c) DWT(135)

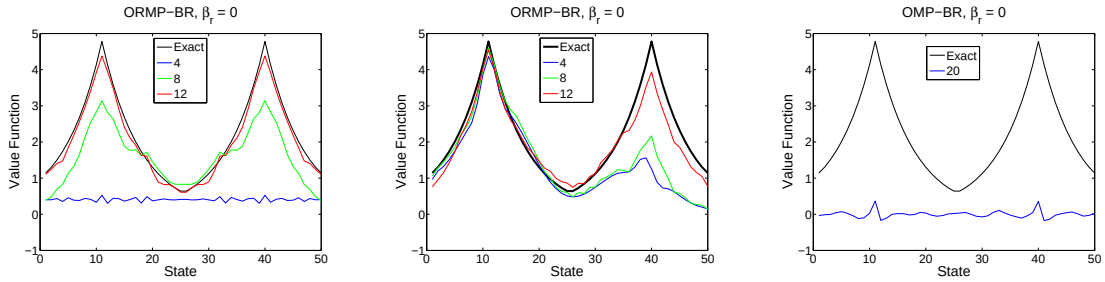
(d) DWT(50) (e) DWT(50) (f) DWT(235)

Figure 6.1. Results of OMP-FP with the PVF and diffusion wavelet dictionaries.

- ORMP-BR

Figure 6.2 shows results using ORMP-BR and OMP-BR with the PVF and diffusion wavelet dictionaries. Interestingly, the only basis selection algorithm that worked in conjunction with the BR least-squares method was ORMP. Notice the approximate value function learned using OMP-BR is very poor (which was also the case for both LASSO-BR and LARS-BR). We show the value function from OMP-BR with 20

basis functions, which is more than enough for an excellent approximation of V^*. On the other hand, ORMP-BR produced excellent approximations when using 8 or 12 basis functions.



(a) PVFs (b) DWT(235) (c) PVFs

Figure 6.2. Results of ORMP-BR and OMP-BR with different dictionaries.

- OMP-H_2 with the PVF dictionary

  Figure 6.3 shows results using OMP-H_2 and LASSO-H_2 with the PVF dictionary. Intermediate values of ξ between 0 and 1 tend to produce approximate value functions between the extremes produced by the FP and BR algorithms.

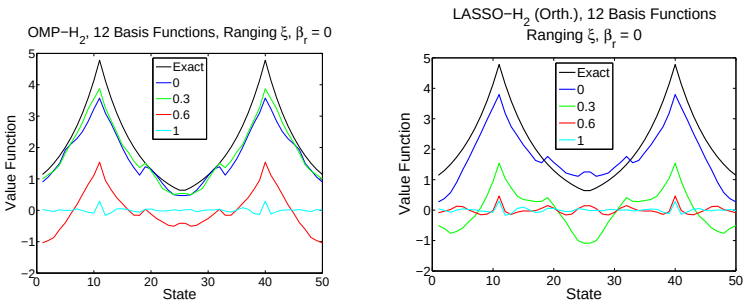


Figure 6.3. Results of OMP-H_2 and LASSO-H_2 with the PVF dictionary using 12 basis functions while varying ξ ($\xi = 0$ is equivalent to FP and $\xi = 1$ is equivalent to BR).

- LASSO-FP

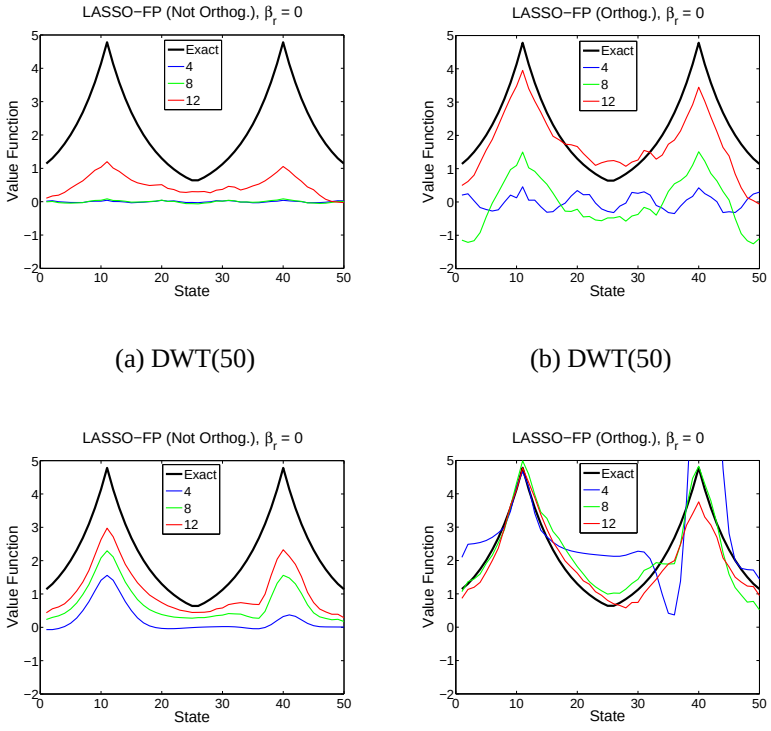
  The LASSO-FP algorithm performed very differently depending on whether the dictionary was orthogonal (PVFs and DWT(50)) or overcomplete (DWT(135) and DWT(235)). Figure 6.4 shows the results using LASSO-FP both with and without the optional orthogonalization step at the end of Algorithm 9. The magnitude of the approximate value function without the orthogonalization step was very small when using the orthogonal dictionaries. This occurs because the LASSO algorithm, which is conservative in its setting of the coefficients $w_{\mathcal{I}}$ by design, moves in an equiangular direction amongst orthogonal elements $\Phi_{\mathcal{I}}$. When the elements $\Phi_{\mathcal{I}}$ are not orthogonal, as in the results with DWT(235), adjusting the coefficient vector $w_{\mathcal{I}}$ can lead to larger steps in approximating the value function.

  When the orthogonalization step in Algorithm 9 is used (which means the LASSO-FP algorithm is used just for basis selection, not for setting the coefficients), the magnitude of the approximate value functions naturally becomes larger. The approximate value functions were very accurate when 8 and 12 basis functions were used from the dictionary DWT(235).

  Note we do not show results using LARS-FP because they are nearly identical, and in some instances exactly identical, to LASSO-FP.

- Indirect scheme with an orthogonal dictionary

  The experiments in this section were conducted using Algorithm 7 under three conditions. First, the while loop in Algorithm 7 was executed for 10 iterations. Second, we used a single termination criterion for the basis selection algorithm. The algorithm stopped when it had selected a specified number of basis functions. Third, we always used the optional third step in Algorithm 7 which is to set the weights on the selected features using a least-squares method. We used the BR and FP least-squares



(a) DWT(50) (b) DWT(50)

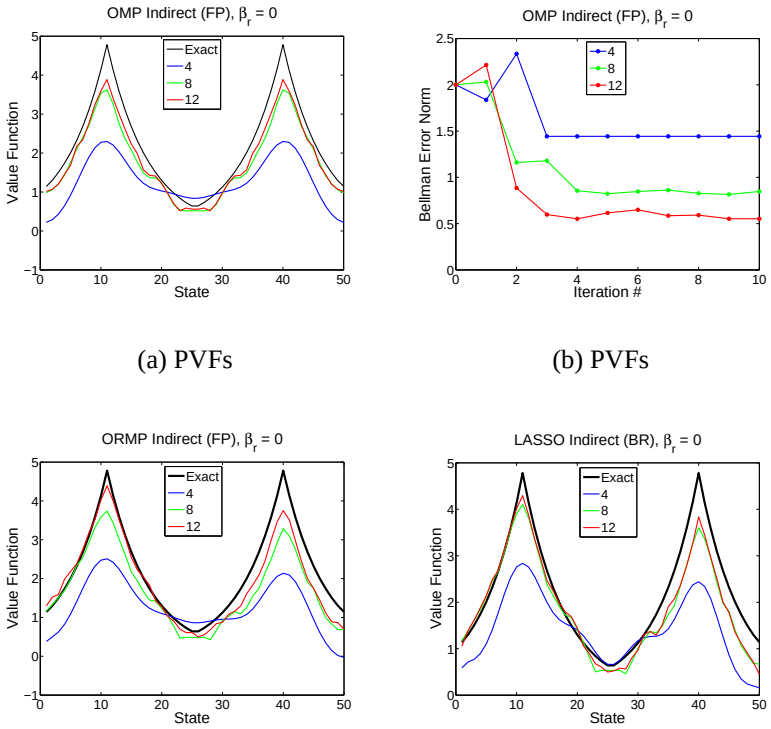
(c) DWT(235) (d) DWT(235)

Figure 6.4. Results of LASSO-FP using diffusion wavelet dictionaries. The value functions are shown with and without the (optional) orthogonalization step in Algorithm 9.

methods. Since BR and FP produced similar results, we do not report results using the hybrid method H_2.

The indirect scheme with an orthogonal dictionary (both PVFs and DWT(50)) produced accurate approximate value functions for all basis selection methods (OMP, ORMP, LASSO, LARS) and both the FP and BR least-squares methods. Figure 6.5 shows results using the OMP and ORMP algorithms with FP and the LASSO algorithm with BR. For the OMP algorithm with FP, there is also a plot of the Bellman error norm $\|T^{\pi}(\Phi_{\mathcal{I}} w_{\mathcal{I}}) - \Phi_{\mathcal{I}} w_{\mathcal{I}}\|^2$ after each iteration of Algorithm 7. We just show the Bellman error plots for the OMP algorithm to point out that the Bellman error is not monotonically decreasing. The Bellman error plots for ORMP and LASSO were very similar to those for OMP.

The resulting value functions were noticeably better than those produced using the direct scheme for approximate policy evaluation. The difference is easily recognized by looking at the value functions estimated using 4 basis functions. Most of the results using the direct scheme produced very poor approximations with just 4 basis functions. But the results were quite good when using the indirect scheme. This supports our hypothesis that the direct policy evaluation scheme can limit the efficacy of the basis selection algorithm by forcing it to follow the Neumann series.



(a) PVFs (b) PVFs

(c) DWT(50) (d) DWT(50)

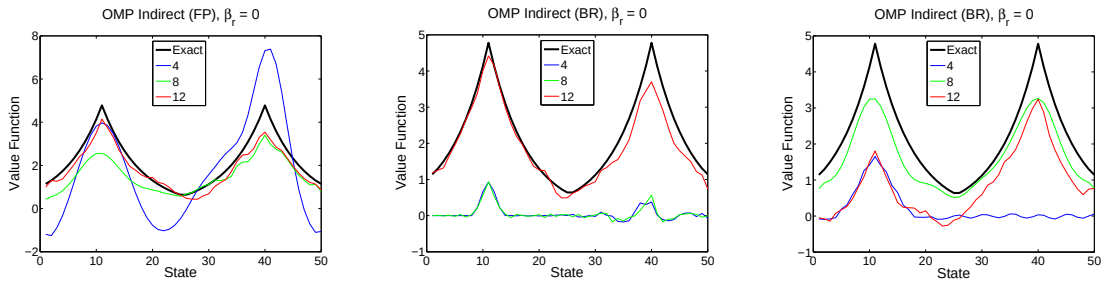
Figure 6.5. Results using the indirect policy evaluation scheme with orthogonal PVF and diffusion wavelet dictionaries.

- Indirect scheme with an overcomplete dictionary

  Figure 6.6 shows the results using the indirect policy evaluation scheme with overcomplete diffusion wavelet dictionaries. Since all basis selection algorithms performed similarly, we just show plots for the OMP algorithm. This is done for both

the FP and BR least-squares methods. Overall, the results using FP were better than those using BR (especially when fewer basis functions were used).

The approximate value functions are less smooth than those produced using orthogonal dictionaries. The results with only 4 basis functions are significantly worse than when 4 basis functions are used from an orthogonal dictionary.



(a) DWT(235) (b) DWT(235) (c) DWT(135)

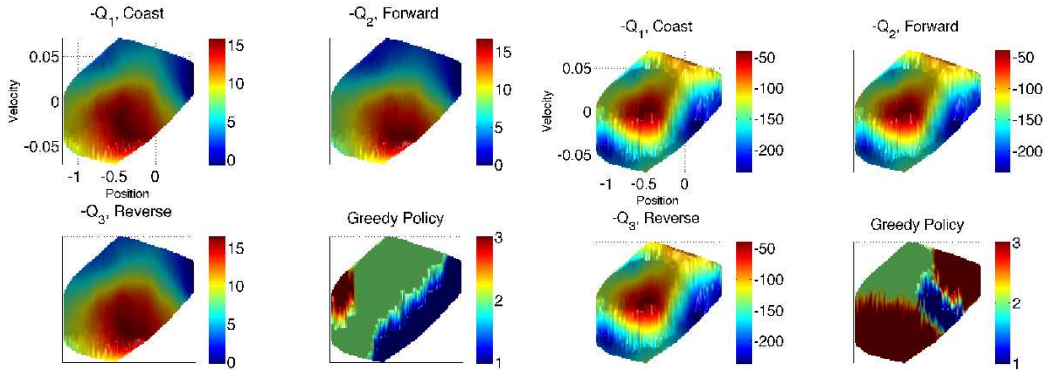
Figure 6.6. Results using the indirect policy evaluation scheme with overcomplete diffusion wavelet dictionaries.

6.4.2 Approximate Policy Iteration

The simple policy evaluation experiments in the previous section were completed using the MDP model and evaluating just a single policy. In this section, we extend beyond this idealized setting to the full approximate policy iteration problem where a policy must be learned from a fixed set of samples. Furthermore, we combine all three components of the dissertation (regularized hybrid least-squares algorithms, efficient basis construction algorithms for graphs, and the basis selection methods) into a single combined architecture. We try to provide intuition as to how these different components interact to produce a final approximate value function and corresponding policy.

Experiments were conducted on the mountain car task using samples from 100 episodes, each of at most 100 steps, of a random policy. The results from Chapter 5 (Figure 5.7) on this domain showed that it was possible to learn policies that could reach the goal, albeit

not optimally, without performing basis selection. This was done using the 20 smoothest Laplacian eigenvectors as a basis. To make the problem somewhat more complex, we restrict the algorithm to only 8 basis functions computed using the AMLS algorithm. With this limited representational capacity, neither the fixed point nor Bellman residual least-squares algorithms were able to learn a policy that reliably attains the goal. It is instructive, however, to see what type of action-value functions these methods do learn. Figure 6.7 shows the action-value functions and corresponding greedy policies for both the BR and FP least-squares methods.[9] There are four plots per method. The first three plots are the action-value functions for the actions coast, forward, and reverse. The fourth plot shows the greedy policy attained from the action-value functions (where the color corresponds to the action). All of these plots are shown from a top-down view of the two dimensional state space. To keep the figures legible, we only show the axis labels for the first of the four plots; the remaining three plots are on the same scale.



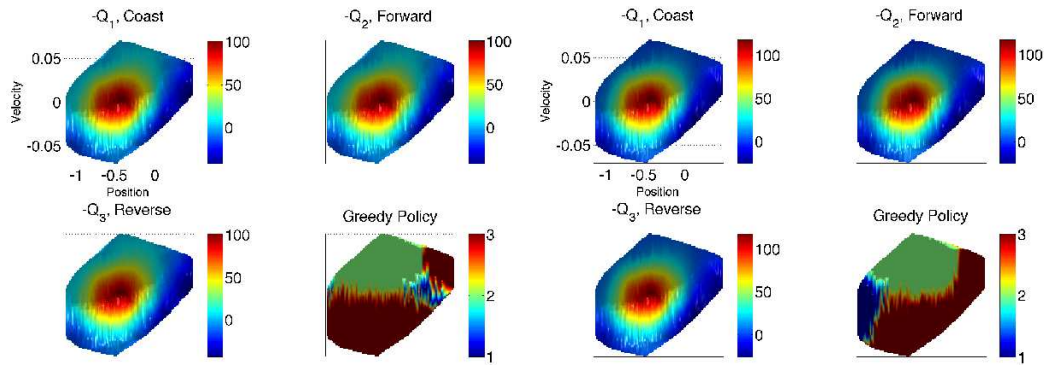
(a) Bellman Residual (b) Fixed Point

Figure 6.7. Action-value functions and policies learned using the BR and FP least-squares policy iteration algorithms and using the 8 smoothest Laplacian eigenvectors (computed using the AMLS algorithm) as a basis.

[9] Strictly speaking, we plot the *negative* of the action-value function as is customary in the RL literature.

There are two interesting things to point out about the action-value functions in Figure 6.7. For the BR least-squares algorithm, notice that the action-value functions are very smooth and that the values range from 0 to -15. The action-value functions learned using the FP method have a much larger gradient. Further, the values range from 50 to 200. That the maximum value is 200 (instead of 0) is less relevant than the fact the spread of values is 150 versus a spread of just 15 for the action-value functions learned using the BR method. This indicates the BR algorithm, via its objective function that minimizes the Bellman residual norm, is *constraining* the range of values as opposed to the basis functions not being expressive enough. This leads to the hypothesis that if the FP method were appropriately regularized, then it could compute an accurate action-value function using these same 8 basis functions. To test this hypothesis, we evaluated two ideas. First, we added Laplacian-based regularization ($\beta_r = 0.1$) to the FP method. Second, we used the hybrid least-squares algorithm with an intermediate weighting ($\xi = 0.5$) to enforce some penalty for having a large Bellman residual norm. Both ideas resulted in better action-value functions and better policies. Starting from the typical start state at the bottom of the hill, the goal is reached in 160 steps (on average) for the policy from the FP method with Laplacian regularization and 219 steps for the policy from the hybrid method (the results were even better - 130 steps to goal - when the hybrid method was used with Laplacian regularization). The action-value functions and greedy plots are shown in Figure 6.8. Notice the range of values for the action-value functions is more in line with the optimal action-value function's range.

This is an interesting result that captures the idea behind the hybrid least-squares algorithm. By placing some weight on minimizing the Bellman residual norm, hybrid methods in effect regularize the solutions produced by the fixed point methods. One can argue this is a more natural form of regularization for MDPs than using graph-based regularization (since it stems from the Bellman equation), but on this task both forms of regularization



(a) Fixed Point, Laplacian Regularization

(b) Hybrid H_2

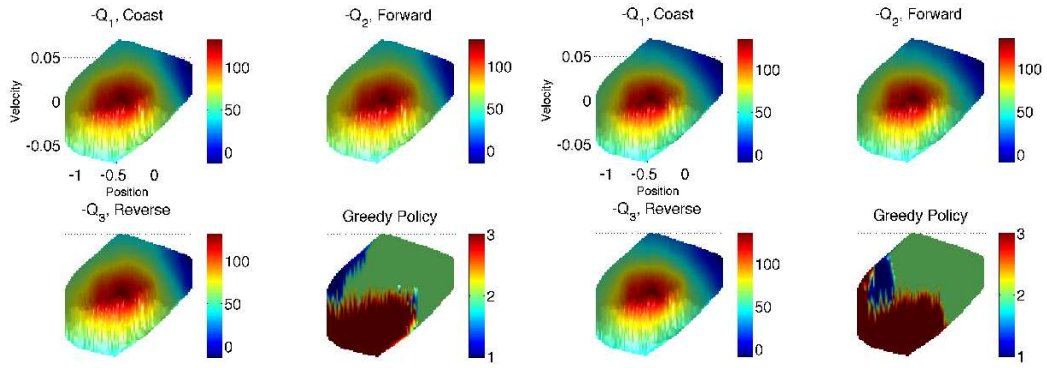
Figure 6.8. Action-value functions and policies learned using approximate policy iteration with the FP method including Laplacian-based regularization ($\beta_r = 0.1$) and the hybrid H_2 method ($\xi = 0.5$).

have a similar effect. We found the results were more sensitive to the Laplacian regularization parameter β_r than to the hybrid parameter ξ.

These results show that even with this limited set of basis functions, it is possible to learn a policy that can reach the goal. However, notice the action-value functions in Figure 6.8 do not accurately capture the optimal action-value function. The only portion of the state space that should have a value close to 0 (which corresponds to dark blue in the plots) is the region in the upper right-hand corner near the goal. The plots show the dark blue values encircle the outside of the state space, which is clearly incorrect. It is interesting to consider whether basis selection algorithms can choose a better subset of basis functions. We still limit the algorithms to 8 basis functions per action, but they are free to select from a dictionary. For this data set, we used a graph containing 700 vertices and computed 100 approximate Laplacian eigenvectors using the AMLS algorithm. The 100 approximate eigenvectors constitute the dictionary.

We limit our presentation of the results to a few interesting cases. First, for the indirect scheme including the optional orthogonalization step in Algorithm 7, the action-value functions were unstable when using the FP least-squares algorithm and any of the basis

selection methods. When using the Bellman residual least-squares algorithm, the action-value functions were too smooth. The hybrid least-squares algorithm, however, resulted in both good policies and accurate action-value functions. Figure 6.9 shows the action-value functions and policies learned using the hybrid least-squares method with OMP and LASSO for basis selection. In particular, notice how the dark blue region of the action-value function plots is confined to just the region near the goal state. This improvement in the representation of the action-value function came as a result of the basis selection algorithms picking elements useful for representing the steep cliff in the mountain car value function. Figure 6.10 shows two such basis functions that the algorithms selected. These are the 12^{th} and 14^{th} smoothest Laplacian eigenvectors in the dictionary.



(a) OMP with H_2 (b) LASSO with H_2

Figure 6.9. Action-value functions and policies learned using the indirect policy iteration scheme with the hybrid least-squares method and the OMP (left) and LASSO (right) algorithms.

We also used the indirect scheme without the orthogonalization step. Note that, without orthogonalization, the indirect scheme in Algorithm 7 is equivalent to Ernst's fitted Q-iteration algorithm [36] with the exception being that the value function in Algorithm 7 is linear in the features. Figure 6.11 shows the action-value function and policy learned using this scheme with LASSO. The approximate action-value function is not close to the optimal action-value function, but its greedy policy is effective (reaching the goal in 131

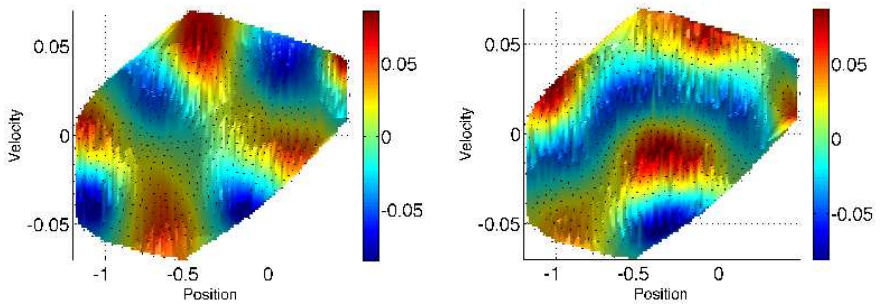


Figure 6.10. The 12th (left) and 14th (right) elements from the Laplacian eigenvector dictionary. The basis selection algorithms chose these elements, which are useful for capturing the steep cliff in the mountain car value function (near the goal region).

steps). Interestingly, although it had the ability to do so, this technique did not change the basis functions from the original 8 smoothest elements in the dictionary.

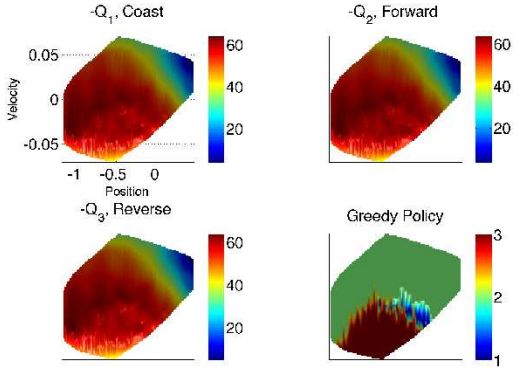


Figure 6.11. The action-value function and greedy policy when using LASSO in the indirect scheme without orthogonalization.

In general, we found the direct schemes for combining approximate policy evaluation and basis selection to be less stable. LASSO-H_2 AND LARS-H_2 produced the best results. This is due to their conservative updates to the basis function coefficients. On the other hand, when OMP-H_2 and ORMP-H_2 select a new feature, the coefficients are set by making the residual completely orthogonal to the selected basis functions. This method is overly aggressive and lead to instability in the action-value function representation. We believe it

is possible to dampen this aggressive behavior by constraining the weights (using Laplacian regularization or a simple L_2 penalty $\|w\|_2$).

We draw two conclusions from the experiments in this section. First, the hybrid least-squares method provides regularization to the approximate value function. This occurs because the hybrid method's objective function includes the Bellman residual norm, which constrains a state to have a similar value to its preceding and succeeding states. We demonstrated the usefulness of the hybrid algorithm, in spite of its single sample bias, both in this section and in Chapter 4. Second, we found that basis selection algorithms can find a better set of basis functions *provided* the policy evaluation algorithm is stable. When the policy evaluation algorithm is unstable, the basis selection algorithms can select a poor set of elements and further exacerbate the problem. Thus, we believe regularization is very important. We selected regularization parameters by hand in this section. In the future, we plan to automate this process.

6.5 Conclusions

Proto-value functions and diffusion wavelets are graph-based basis functions that capture topological structure of the MDP state space. The basis functions are independent of any policy and therefore can be used to approximate any policy's value function. A mechanism is required though to select a subset of the basis functions for approximating a value function. The previous approach to using PVFs and diffusion wavelets used the following basis selection heuristic: the most global functions were selected regardless of the policy being evaluated. This heuristic is simple and leads to smooth approximations, but it does not fully utilize the graph-based dictionaries. To make better use of the dictionaries, a sparse basis selection algorithm must be combined with approximate policy evaluation. We evaluated a scheme that directly combines basis selection and policy evaluation and a scheme that indirectly combines them via an iterative process. Both schemes are general and can be used with any set of basis functions. The hybrid least-squares method was used

for approximate policy evaluation. Specifically, we used the Laplacian-based regularized form of the hybrid algorithm developed in Section 4.4. For the basis selection algorithm, we implemented orthogonal matching pursuit (OMP), order recursive matching pursuit (ORMP), and LASSO and LARS. A systematic study was conducted on a simple chain MDP to determine the most promising way(s) of combining these various components. From these experiments, we summarize with the following four findings.

1. We showed that the direct scheme for sparse approximate policy evaluation, when combined with the fixed point least-squares method, constrains the order in which a basis selection algorithm selects elements from a dictionary. The order is dictated by the elements in the Neumann series, $\sum_{i=0}^{\infty}(\gamma P^{\pi})^{i}R^{\pi}$. This can lead to the selection of basis functions that fit some of the early terms in the series, but are in fact not useful for representing the underlying value function. Of course, an algorithm like LASSO that can prune basis functions has the possibility of removing basis functions that become useless. The indirect scheme for sparse approximate policy evaluation sidesteps this issue by separating the Bellman equation from the basis selection algorithm. This adds computational complexity, but frees up the basis selection algorithm to represent the value function in the order it sees fit.

2. The graph Laplacian, which is used in constructing PVFs and diffusion wavelets, can also be used to provide regularization. Laplacian-based regularization can help smooth out the approximate value function. It also provides a bias toward smoother basis functions in the dictionary. This bias can be helpful when using the direct scheme for sparse approximate policy evaluation. We speculate that in an online setting, it may be beneficial to adjust the amount of regularization over time as more samples are seen.

3. *For direct sparse approximate policy evaluation:*

   The OMP-FP algorithm produced accurate approximations when using an orthonor-

mal dictionary, but became unstable when using an overcomplete dictionary due to matrix $\hat{A}_{\mathcal{I},\mathcal{I}}^{-1}$ becoming nearly singular. The algorithm could be made more robust by checking the condition number of the matrix before including a new basis function. The more conservative nature of LASSO-FP and LARS-FP lead to accurate approximate value functions; however, when using an orthonormal dictionary, these algorithms generated approximate value functions with small magnitude (without the orthogonalization step at the end of Algorithm 9). The only algorithm that worked using the Bellman residual least-squares method was ORMP-BR. This was an interesting result that shows one must be careful when combining basis selection and approximate policy evaluation algorithms.

4. *For indirect sparse approximate policy evaluation:*

   OMP, ORMP, and LASSO all produced accurate approximate value functions while using both the fixed point and Bellman residual least-squares methods. When using a small number of basis functions, the algorithms performed better with an orthogonal dictionary as opposed to an overcomplete dictionary. Overall, the results were noticeably better than using an orthonormal dictionary with the direct scheme for sparse approximate policy evaluation. This provides some evidence for the hypothesis that the indirect scheme can select a more efficient set of basis functions than the direct scheme.

In the approximate policy iteration experiments, policies were learned from a set of samples. The results attained with and without basis selection indicate the importance of regularization. In particular, when changing the basis functions, the Bellman residual should be controlled for basis selection to remain stable. There are multiple ways to ensure stability: graph-based regularization, use of the hybrid least-squares algorithm, and/or use of a conservative basis selection algorithm like LARS/LASSO. Each of these methods helps protect against large Bellman residuals.

The policy evaluation experiments partially demonstrate the expressiveness and flexibility of the diffusion wavelet dictionary. However, we believe the true value of diffusion wavelets will be evident on more challenging value functions with discontinuities and different degrees of smoothness. For future work, it would be worthwhile further decomposing the diffusion wavelet tree using diffusion wavelet packets [18]. This increases the size of the dictionary and provides even more flexibility for function approximation.

The benefit of maintaining a basis function dictionary is the flexibility to approximate many different functions. This benefit comes at the cost of storing a potentially large number of elements in the dictionary; therefore, efficient storage schemes are very important. As an example, recall the Kronecker product method from Chapter 5 stores the dictionary in a compressed format. Parametric methods for representing the dictionary could also prove useful.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

7.1 Summary

In this dissertation, we proposed automatic basis construction algorithms and an accompanying approximate policy evaluation method for Markov decision processes. The ability to automatically learn useful representations is an important and fundamental challenge for an autonomous agent. RL agents must be able to meet this challenge to both deal with the complexity of real world environments and to go beyond their initial representations given by human designers.

Our work builds upon a recently introduced graph-based approach to generating representations [67, 63]. In this application, graphs reflect the geometric structure of a MDP state space. An important attribute of the graph-based approach is that it circumvents the dimensionality of the state space. While a MDP state space may nominally be very high dimensional, if the reachable state space in fact lies on a lower dimensional surface, then the graph-based approach can leverage this information to avoid the "curse of dimensionality." Basis functions generated from these state space graphs are well-suited to represent certain classes of value functions.

The goal of this dissertation was twofold: (1) to scale the graph-based approach to handle larger amounts of data, and (2) to effectively and efficiently use the basis functions to perform approximate policy evaluation. To scale the graph-based approach, we proposed one matrix factorization algorithm and another multiscale algorithm. Both algorithms produce basis functions that approximate the original bases proposed by Mahadevan and Maggioni [67, 63]. Once constructed, the basis functions act as a dictionary. Repre-

senting a particular policy's value function using a linear approximation is accomplished by selecting elements from the dictionary and assigning each element a scalar weight. We evaluated several basis selection algorithms and introduced a hybrid least-squares method for setting the weights. Although we mostly explore using the basis selection algorithms and the hybrid least-squares method with the graph-based basis functions, these algorithms can be applied to other types of basis functions as well. The remainder of this summary covers our contributions in greater detail.

In Chapter 4, we presented hybrid least-squares algorithms for approximate policy evaluation. The term "hybrid" is used to emphasize the fact that the algorithm parametrically combines (and generalizes) two common RL least-squares methods. At one extreme, the hybrid algorithm defaults to minimizing the norm of the Bellman residual (BR). At the other extreme, the norm of the projected Bellman residual is minimized (we refer to this as the fixed point (FP) method).

By using a linear combination of the BR and FP objective functions, hybrid algorithms allow for finding solutions in between those computed by the BR and FP methods. This can be useful when the hybrid least-squares algorithm is used within a policy iteration loop. We do not have a theoretical explanation of this result; rather, empirical results show the hybrid method appears to temper large changes to the value function that the FP method can make between rounds of policy iteration. Experiments in a discrete grid MDP, the challenging problem of Tetris, and mountain car demonstrated that the hybrid algorithm can, in some cases, find better policies. We also proposed a *regularized* hybrid least-squares algorithm which uses the graph Laplacian [20]. The Laplacian penalizes functions that are not smooth according to the structure of the graph. This type of regularization is useful for MDPs when the domain is stochastic and relatively few samples are available for learning a policy.

Two recently introduced approaches to automatically generating basis functions from a MDP state space graph are to form graph Laplacian eigenvectors (proto-value functions [67]) and diffusion wavelets [63]. Computing eigenvectors and diffusion wavelet trees from

large, sparse matrices can be computationally intensive. To scale these basis construction methods to larger graphs and thus larger MDP state spaces, we introduced two algorithms in Chapter 5. The first algorithm is based on matrix factorization using the Kronecker product. The Kronecker product is particularly relevant because it preserves the spectral structure of matrices. In this approach, smaller matrices are automatically computed and combined together via the Kronecker product to approximate the (larger) original matrix. Laplacian eigenvectors or diffusion wavelet trees can then be generated from these smaller matrices. We showed how the Kronecker product method significantly saves both time and memory. Experiments using the basis functions produced by the Kronecker product method were mixed. We attribute this result to the Kronecker product's block structure not allowing for sufficient representation of arbitrary matrices. To overcome this limitation, we proposed using a second algorithm called Automated Multilevel Substructuring (AMLS). AMLS recursively decomposes a matrix into smaller submatrices, computes eigenvectors for the submatrices, and uses those solutions to approximate eigenvectors of the original matrix. We proved the algorithm is applicable to graph Laplacian matrices. The recursive nature of AMLS allows for a very fast parallel implementation. The algorithm can handle graphs up to one hundred times larger than standard eigensolvers can manage (given equal computing resources). Aside from its scalability, we also demonstrated the basis functions computed by AMLS performed just as well in policy iteration experiments as those computed using exact eigensolvers.

In Chapter 6, we evaluated four well-established basis selection algorithms: orthogonal matching pursuit [82], order recursive matching pursuit [75], the LASSO [103], and least angle regression [33]. Basis selection algorithms choose as few elements as possible from a dictionary in order to represent a value function. In tailoring the representation to a particular value function, selection algorithms provide flexibility and computational efficiency. We employed these algorithms using graph-based basis functions as a dictionary. Other

types of basis functions could also be used with the selection algorithms. Our work [45] is the first time Laplacian eigenvectors and diffusion wavelets have been used in this manner.

Along with choosing elements from the dictionary, basis selection algorithms must also assign weights to the basis functions. This was accomplished using the regularized hybrid least-squares method developed in Section 4.4. We evaluated two different ways of combining the policy evaluation method and the basis selection algorithm. The distinction between the two ways is whether the policy evaluation method is directly encoded in the basis selection algorithm. We showed this distinction can have a significant effect on how the dictionary is utilized. Interestingly, our experiments showed that the basis selection algorithms perform differently depending on whether the dictionary consists of an orthonormal or overcomplete set of basis functions. When the dictionary is overcomplete (as is the case with the complete diffusion wavelet tree), the conservative nature of the LASSO and least angle regression algorithms proved more useful than the aggressive matching pursuit methods.

7.2 Future Work

There are a number of interesting directions for future work.

- Learning algorithms

  The hybrid least-squares algorithm presented in Chapter 4 requires setting a scalar parameter to a value between 0 and 1. For Baird's incremental version of the hybrid algorithm [3], he proposed setting this parameter to guarantee convergence. Since ensuring convergence is unnecessary for the least-squares version, we have more flexibility. In the experiments, we selected a particular value and held it fixed throughout the policy iteration loop. In future work, we would like to provide a framework for automatically setting the parameter's value. This should be done separately for each round of policy iteration. One of the factors determining the parameter's value should be the impact of the Bellman residual method's bias. We showed that the bias

is linearly impacted by the parameter (i.e. setting the hybrid method's parameter to 0 causes the solution to be unbiased).

We used least-squares algorithms for approximate policy evaluation because of their data efficiency and because they do not require setting a step-size parameter. However, it is interesting to consider other methods and how they might impact the basis construction and selection problems. We suggest one possibility that focuses more on policies than on value functions. To motivate this change from value functions to policies, we mention an interesting example from the Tetris domain. Using 22 hand-coded basis functions defined in [12], Szita and Lörincz [100] showed that the cross-entropy method (which searched directly for a set of 22 coefficients resulting in good policies) can learn policies that score *100 times better* than policies learned using the same 22 basis functions and a temporal difference algorithm.

The least-squares algorithms minimize different functions of the Bellman residual. The rationale for doing so is based on the fact that the expected Bellman residual is 0 for the exact value function. An alternative to this approach is to try to represent the greedy policy associated with the exact value function rather than representing the exact value function itself. This idea was explored in a few different contexts [105, 10, 107], but the main theme uniting this work is to have the algorithm learn the relative value of a state (which is what determines the policy) as opposed to the absolute value of a state. This type of algorithm may make the basis construction and selection problems easier since representing a policy may be simpler than representing a value function. In effect, the algorithm can make larger errors in the Bellman residual as long as it orders the states correctly. We believe this is an interesting area for future work.

- Basis selection and diffusion wavelets

  The basis selection framework presented in Chapter 6 in conjunction with an expressive, overcomplete dictionary like the diffusion wavelet dictionary provides a powerful tool for value function approximation. There are three immediate ways to extend this work. First, the main component, and thus bottleneck, in constructing a diffusion wavelet tree is the sparse QR decomposition algorithm. The QR decomposition algorithm is used to compute both the scaling functions and the wavelet functions. A faster implementation of the algorithm is needed. Also, to scale up, it may be beneficial to approximate the QR decomposition. This might result in the loss of orthogonality between the scaling and wavelet functions at each level of the tree, but that might not be a crucial factor when approximating a value function. The second extension is to exploit the structure of the diffusion wavelet tree when performing basis selection. Our current implementation simply takes each element in the tree, which is stored in a compressed format, and unrolls the element back to an uncompressed format before using the basis selection algorithm. This is inefficient. A faster implementation here can significantly affect the runtime since basis selection occurs each time a new policy is evaluated. The third extension is to explore the use of diffusion wavelet packets [18]. Diffusion wavelet packets allow for splitting the wavelet spaces into an orthogonal sum of smaller subspaces. This creates a larger number of elements in the diffusion wavelet tree which are more localized and thus offer greater flexibility in terms of function approximation.

- Graph construction

  An advantage of the graph-based approach to basis construction in Markov decision processes is its flexibility. As we we have demonstrated in this dissertation, the approach is amenable to both discrete and continuous state spaces. The graphs we constructed from MDP samples were simply based on a user-specified distance

function. One obvious extension is to automatically learn the distance function based on actual transitions. In fact, it is possible to learn several distance functions where each one is responsible for a different portion of the state space. Bowling et al. [15] proposed a simple method accomplishing this.

While the user has the ability to specify the distance function that creates the graph from samples, this is typically done just based on the dynamics of the domain. An interesting extension would be to form the graph not solely based on structural similarity but also based on the type of value functions the learning algorithm is likely to encounter. For example, if two states that are topologically close but in fact have different values for many policies, then the edge weight between these two states in the graph can be decreased. Decreasing the edge weight in turn alters the shape of the basis functions generated from the graph. We proposed an ad-hoc method for adjusting edge weights based on the Bellman residual [47]. This earlier work was a proof of concept, but in the future we hope to determine a more principled approach that is also scalable.

- Instance-based representations

  In order to use the graph-based approach to basis construction for MDPs, the samples/states forming the vertices of the graph must be stored. In other words, the graph and its associated features are an instance-based representation. This should be contrasted with representations using a fixed number of parameters, such as neural networks with a prescribed connectivity. While we have proposed methods for dealing with large graphs, scalability is a concern for any algorithm using an instance-based representation. This issue is not unique to reinforcement learning. Indeed, this is an issue with any kernel method (note the graph Laplacian induces a reproducing kernel Hilbert space [92] and can be considered a kernel method). Understanding the practical limitations of instance-based representations and if and how these limitations can

be circumvented is an ongoing area of research in machine learning. One interesting possibility is to see if instance-based features can be stored more compactly using a parametric representation.

7.3 Final Remarks

A hallmark of human-level intelligence is the ability to successfully perform many tasks across a wide range of environments. In order for autonomous agents to approach this level of flexibility, they must be able to adapt their internal representations of the environments in which they reside. The graph-based methods [67, 63] we considered in this dissertation provide one way to generate flexible representations that capture structural information about an environment. In the reinforcement learning paradigm, an autonomous agent forms such representations and then uses them to learn how to act. We addressed these two interrelated aspects in the context of value function approximation:

1. Given a set of samples from an environment, can we generate graph-based features associated with the samples in a manner that scales well?

2. Given a set of samples from an environment and the graph-based features, how should the features be used to compute a (good) policy?

The set of features dictates the space of approximate value functions that can be represented. The algorithm utilizing the features determines how a policy will be found. We believe a thorough understanding of the confluence of these two areas, automatic feature construction and feature utilization, is an interesting and worthwhile topic for continued RL research.

APPENDIX A

DOMAINS

A.1 Chain MDP

We used the 50 state chain MDP described by Lagoudakis and Parr [56]. Figure A.1 shows a depiction of the problem and the optimal value function. There are 50 discrete states $\{s_i\}_{i=1}^{50}$ and two actions moving the agent left ($s_i \rightsquigarrow s_{\max(i-1,1)}$) and right ($s_i \rightsquigarrow s_{\min(i+1,50)}$). The actions succeed with probability 0.9; failed actions move the agent in the opposite direction. The discount factor is $\gamma = 0.9$. The agent receives a reward of $+1$ when in states s_{10} and s_{41}. All other states have a reward of 0.

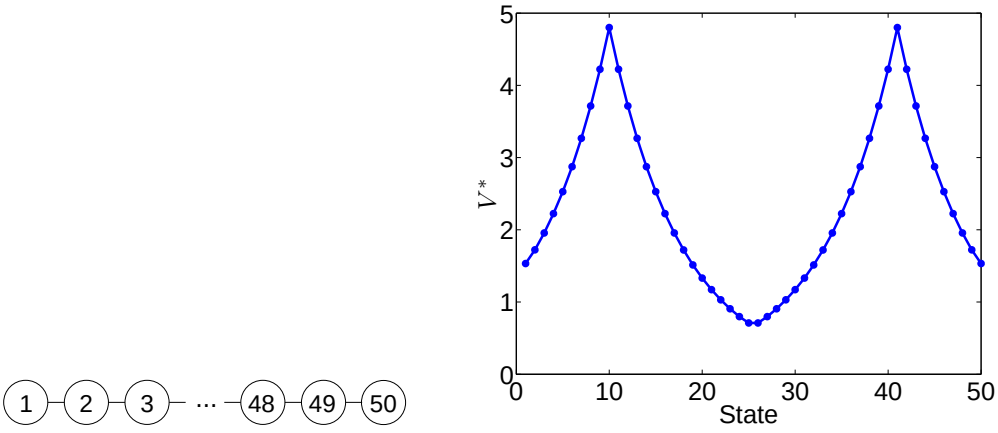


Figure A.1. The chain MDP and the optimal value function.

A.2 Grid MDP

Grid MDPs are simply two dimensional versions of the aforementioned chain MDP. A simple square grid and a two-room grid with one state adjoining the two rooms are shown

in Figure A.2. There are four canonical actions that move the agent up, down, left, or right. The actions succeed with probability 0.9. Unsuccessful actions result in a transition in one of the other three directions (with equal probability). Episodes begin in a random state in the MDP. The discount factor is assumed to be $\gamma = 0.95$ unless otherwise stated. The reward function is 0 except for a few goal states (which are specified on an individual basis for each grid MDP used throughout this dissertation) that have a positive reward.

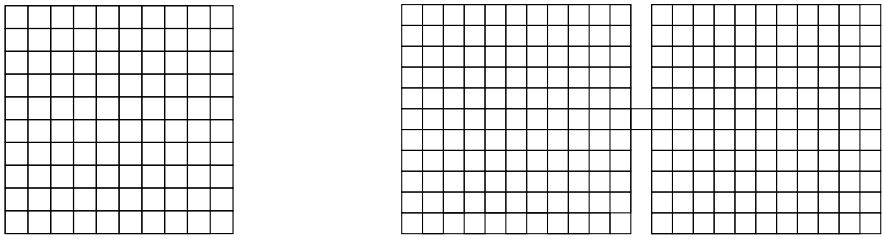


Figure A.2. A 10×10 grid MDP and a two-room version with a single "hallway" state.

A.3 Inverted Pendulum

The inverted pendulum problem requires balancing a pendulum by applying force to the cart to which the pendulum is attached. We used the implementation described by Lagoudakis and Parr [56]. The state space is defined by two variables: θ, the vertical angle of the pendulum, and $\dot{\theta}$, the angular velocity of the pendulum. The three discrete actions are applying a force of -50, 0, or 50 Newtons. Uniform noise from -10 and 10 is added to the chosen action. State transitions are described by the following nonlinear equation

$$\ddot{\theta} = \frac{g \sin(\theta) - \alpha m l \dot{\theta}^2 \sin(2\theta)/2 - \alpha \cos(\theta) a}{4l/3 - \alpha m l \cos^2(\theta)},$$

where a is the noisy control signal, $g = 9.8 m/s^2$ is gravity, $m = 2.0$ kg is the mass of the pendulum, $M = 8.0$ kg is the mass of the cart, $l = 0.5$ m is the length of the pendulum, and $\alpha = 1/(m + M)$. The simulation time step is set to 0.1 seconds. The agent is given a

reward of 0 as long as the absolute value of the angle of the pendulum does not exceed $\pi/2$, otherwise the episode ends with a reward of -1. The discount factor was set to $\gamma = 0.9$. Episodes begin with both state variables at value 0.

A.4 Mountain Car

The task in the mountain car domain is to drive an underpowered vehicle, situated in a valley, to the top of the mountain on the right [98]. Figure A.3 shows a depiction of the problem. There are two state variables: the position (x) and velocity ($\dot{x}$) of the car. There are three actions corresponding to a positive ($a = 1$), negative ($a = -1$), and zero ($a = 0$) force. The equations of motion are:

$$\dot{x}_{t+1} = \text{bound}[\dot{x}_t + 0.001a_t - 0.0025\cos(3x_t)]$$
$$x_{t+1} = \text{bound}[x_t + \dot{x}_{t+1}]$$

where the bound operation ensures $-1.2 \leq x_{t+1} \leq 0.5$ and $-0.07 \leq \dot{x}_{t+1} \leq 0.07$. The velocity $\dot{x}_{t+1}$ is reset to 0 when the position x_{t+1} becomes less than -1.2. When the position exceeds 0.5, the car has reached the top of the hill on the right and the episode is terminated. The reward for reaching the goal is 0; every step where the goal is not achieved results in a reward of -1. The discount factor is $\gamma = 0.99$. Episodes begin in a state with $x_1 = -0.5$ and $\dot{x}_1$ randomly selected from the set $[-0.07, -0.06, \ldots, 0.06, 0.07]$. The distribution over $\dot{x}_1$ allows for easy exploration of the state space.

A.5 Acrobot

The acrobot [98] is an underactuated double pendulum. This is an interesting and well-studied problem due to the nonlinearity of the dynamics. It consists of two links where torque can only be applied at the second joint (Figure A.4). The system is described by four continuous variables: the two joint angles, θ_1 and θ_2, and the angular velocities, $\dot{\theta}_1$

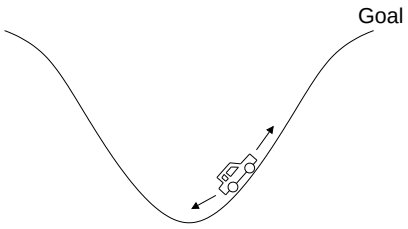


Figure A.3. The mountain car domain.

and $\dot{\theta}_2$. There are three actions corresponding to positive ($a = 1$), negative ($a = -1$), and zero ($a = 0$) torque. We use the same equation of motions and problem parameters as described in Chapter 11.3 of [98]. The time step was set to 0.05 and actions were selected after every fourth update to the state variables according to the equations of motion [97]. The goal for this domain is to raise the tip of the second link above a certain height in minimum time (we used a height of 1, where both links have a length of 1). The reward function is therefore -1 for each time step until the goal is achieved and the discount factor is $\gamma = 0.99$. Episodes begin with the all state variables at value 0 which corresponds to the two links hanging straight down and motionless.

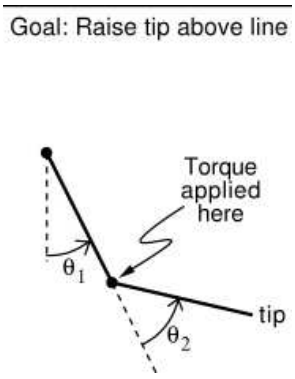


Figure A.4. The acrobot domain.

A.6 Tetris

The game of Tetris was designed by Alexey Pajitnov in 1985. It consists of a board with 20 rows and 10 columns as shown in Figure A.5. Puzzle pieces, each containing four blocks in different positions, fall vertically down the board. The player's objective is to orient the piece as it is falling to create a horizontal row of blocks with no gaps. When this is accomplished, the completed row disappears and any blocks above the row fall down. The game ends when a block is placed in the top row, not allowing further game pieces to enter the board.

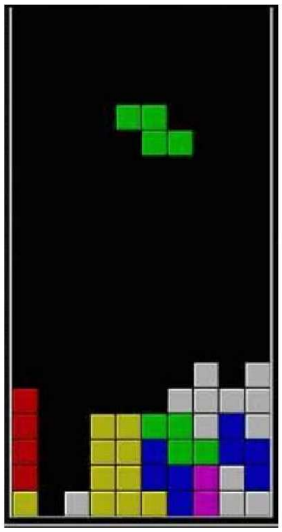


Figure A.5. The Tetris domain.

APPENDIX B

PARTITIONED MATRIX INVERSE

The OMP-H_2, ORMP-H_2, LASSO-H_2, and LARS-H_2 algorithms in Section 6.2.1 formed the matrix $\hat{A}_{\mathcal{I},\mathcal{I}}$ and vector $\hat{b}_{\mathcal{I}}$. Each algorithm then inverts the matrix $\hat{A}_{\mathcal{I},\mathcal{I}}$. This is very wasteful when the active set $\mathcal{I}$ only changes by one element at a time. To take advantage of the single element insertion and removal, $\hat{A}^{-1}_{\mathcal{I},\mathcal{I}}$ can be incrementally formed using the following partitioned matrix inverse property. Consider a square matrix A' partitioned as follows:

$$A' = \begin{bmatrix} A & b \\ c^T & d \end{bmatrix}$$

where matrix A is square, b and c are vectors, and d is a scalar. Then the inverse of A' can be computed from the inverse of A as:

$$A'^{-1} = e \begin{bmatrix} (e^{-1}A^{-1} + A^{-1}bc^TA^{-1}) & -A^{-1}b \\ -c^TA^{-1} & 1 \end{bmatrix}$$

where $e = (d - c^TA^{-1}b)^{-1}$. Computing A'^{-1} in this manner has quadratic complexity instead of cubic. OMP-H_2, ORMP-H_2, LASSO-H_2, and LARS-H_2 can exploit this property by maintaining the matrix $\hat{A}^{-1}_{\mathcal{I},\mathcal{I}}$. When inserting a new element j^* into $\mathcal{I}$, the update is as follows:

$$
\begin{aligned}
\mathcal{I} &\leftarrow \mathcal{I} \cup \{j^*\} \\
\hat{A}^{-1}_{\mathcal{I},\mathcal{I}} &\leftarrow \begin{bmatrix} (\hat{A}^{-1}_{\mathcal{I},\mathcal{I}} + u_{j^*}\hat{A}^{-1}_{\mathcal{I},\mathcal{I}}\hat{A}_{\mathcal{I},j^*}\hat{A}_{j^*,\mathcal{I}}\hat{A}^{-1}_{\mathcal{I},\mathcal{I}}) & -u_{j^*}\hat{A}^{-1}_{\mathcal{I},\mathcal{I}}\hat{A}_{\mathcal{I},j^*} \\ -u_{j^*}\hat{A}_{j^*,\mathcal{I}}\hat{A}^{-1}_{\mathcal{I},\mathcal{I}} & u_{j^*} \end{bmatrix} \\
\hat{b}_{\mathcal{I}} &\leftarrow \begin{bmatrix} \hat{b}_{\mathcal{I}} \\ \hat{b}_{j^*} \end{bmatrix},
\end{aligned}
$$

where:

$$
\begin{aligned}
u_{j^*} &\leftarrow (\hat{A}_{j^*,j^*} - \hat{A}_{j^*,\mathcal{I}}\hat{A}^{-1}_{\mathcal{I},\mathcal{I}}\hat{A}_{\mathcal{I},j^*})^{-1} \\
\hat{A}_{j^*,j^*} &\leftarrow \sum_{i=1}^{n} \rho(s_i)\left[\phi_{j^*}(s_i)(\phi_{j^*}(s_i) - \gamma\phi_{j^*}(s_i')) + \beta_r g_{j^*}(s_i)g_{j^*}(s_i)\right] \\
\hat{A}_{\mathcal{I},j^*} &\leftarrow \sum_{i=1}^{n} \rho(s_i)\left[\phi_{\mathcal{I}}(s_i)(\phi_{j^*}(s_i) - \gamma\phi_{j^*}(s_i')) + \beta_r g_{\mathcal{I}}(s_i)g_{j^*}(s_i)\right] \\
\hat{A}_{j^*,\mathcal{I}} &\leftarrow \sum_{i=1}^{n} \rho(s_i)\left[\phi_{j^*}(s_i)(\phi_{\mathcal{I}}(s_i) - \gamma\phi_{\mathcal{I}}(s_i'))^T + \beta_r g_{j^*}(s_i)g_{\mathcal{I}}(s_i)^T\right] \\
\hat{b}_{j^*} &\leftarrow \sum_{i=1}^{n} \rho(s_i)\phi_{j^*}(s_i)r_i.
\end{aligned}
$$

Similarly, when LASSO-FP removes an element $j^{\#}$ from $\mathcal{I}$, the matrix $\hat{A}^{-1}_{\mathcal{I},\mathcal{I}}$ can be shrunk with the following update:

$$
\begin{aligned}
\mathcal{I} &\leftarrow \mathcal{I} - \{j^{\#}\} \\
\text{Partition the current } \hat{A}^{-1}_{\mathcal{I},\mathcal{I}} &\leftarrow \begin{bmatrix} U & x_{j^{\#}} \\ y^T_{j^{\#}} & z_{j^{\#}} \end{bmatrix} \text{ to isolate the influence of } j^{\#} \\
\hat{A}^{-1}_{\mathcal{I},\mathcal{I}} &\leftarrow U - x_{j^{\#}}y^T_{j^{\#}}/z_{j^{\#}}.
\end{aligned}
$$

APPENDIX C

BASIS SELECTION PSEUDOCODE

The ORMP-H_2 algorithm was described in Section 6.2.1.1. Pseudocode for ORMP-H_2 is shown below in Algorithm 8.

Algorithm 8: ORMP-H_2 with Laplacian-based Regularization

Input: $\{s_i, r_i, s'_i\}_{i=1}^n$, samples generated using policy π
$\phi : S \rightarrow \mathbb{R}^K$, basis function
$\rho : S \rightarrow \mathbb{R}^+$, weighting over the states
$\xi \in [0, 1]$, hybrid parameter ($\xi = 0$ is FP, $\xi = 1$ is BR)
L, graph Laplacian defined over states $\{s_i\}_{i=1}^n$ (graph edges denoted with $\sim$)
$\gamma \in [0, 1]$, discount factor
$\beta_r \in \mathbb{R}^+$, Laplacian-based regularization parameter
$k' \leq K$, maximum allowable number of basis functions

Output: $\mathcal{I}$, set of selected basis functions (indices into ϕ)
$w_{\mathcal{I}}$, weight vector such that $\hat{V}(s) = \phi_{\mathcal{I}}(s)^T w_{\mathcal{I}}$

Initialize active set $\mathcal{I} \leftarrow \emptyset$

while ($|\mathcal{I}| < k'$) and (Bellman residual not converged) **do**

1. Find best inactive element:

$$j^* \leftarrow \text{argmax}_{j \notin \mathcal{I}} \left(\hat{b}_{\mathcal{I}_{+j}}^T \hat{A}_{\mathcal{I}_{+j}, \mathcal{I}_{+j}}^{-1} \hat{b}_{\mathcal{I}_{+j}} \right)$$

where: $\mathcal{I}_{+j} \leftarrow \mathcal{I} \cup \{j\}$

$$\hat{b}_{\mathcal{I}_{+j}} \leftarrow \sum_{i=1}^n \rho(s_i)(\phi_{\mathcal{I}_{+j}}(s_i) - \xi\gamma\phi_{\mathcal{I}_{+j}}(s'_i))r_i$$

$$\hat{A}_{\mathcal{I}_{+j}, \mathcal{I}_{+j}} \leftarrow \sum_{i=1}^n \rho(s_i)[(\phi_{\mathcal{I}_{+j}}(s_i) - \xi\gamma\phi_{\mathcal{I}_{+j}}(s'_i))(\phi_{\mathcal{I}_{+j}}(s_i) - \gamma\phi_{\mathcal{I}_{+j}}(s'_i))^T + \ldots \beta_r g_{\mathcal{I}_{+j}}(s_i) g_{\mathcal{I}_{+j}}(s_i)^T]$$

where: $g(s_i) \leftarrow L(s_i, s_i)\, \phi(s_i)$

$$g(s_i) \leftarrow g(s_i) + L(s_i, s_{nbr})\, \phi(s_{nbr}), \qquad \forall \{s_{nbr} | s_{nbr} \neq s \ \wedge \ s \sim s_{nbr}\}$$

2. Adjust active set:
$\mathcal{I} \leftarrow \mathcal{I} \cup \{j^*\}$
3. Compute $\hat{A}_{\mathcal{I},\mathcal{I}}$ and $\hat{b}_{\mathcal{I}}$:
$\hat{A}_{\mathcal{I},\mathcal{I}} \leftarrow \sum_{i=1}^n \rho(s_i)[(\phi_{\mathcal{I}}(s_i) - \xi\gamma\phi_{\mathcal{I}}(s'_i))(\phi_{\mathcal{I}}(s_i) - \gamma\phi_{\mathcal{I}}(s'_i))^T + \beta_r g_{\mathcal{I}}(s_i) g_{\mathcal{I}}(s_i)^T]$
$\hat{b}_{\mathcal{I}} \leftarrow \sum_{i=1}^n \rho(s_i)(\phi_{\mathcal{I}}(s_i) - \xi\gamma\phi_{\mathcal{I}}(s'_i))r_i$
4. Compute least-squares weights:
$w_{\mathcal{I}} \leftarrow \hat{A}_{\mathcal{I},\mathcal{I}}^{-1} \hat{b}_{\mathcal{I}}$

end while

The LASSO-H_2 and LARS-H_2 algorithm were described in Section 6.2.1.1. Algorithm 9 shows the pseudocode implementing LASSO-H_2 and LARS-H_2.

Algorithm 9: `LARS-H2/LASSO-H2 with Lap.-based Regularization`

Input: $\{s_i, r_i, s'_i\}_{i=1}^n$, samples generated using policy π
$\phi : S \to \mathbb{R}^K$, basis function
$\rho : S \to \mathbb{R}^+$, weighting over the states
$\gamma \in [0, 1]$, discount factor
$\xi \in [0, 1]$, hybrid parameter ($\xi = 0$ is FP, $\xi = 1$ is BR)
L, graph Laplacian defined over states $\{s_i\}_{i=1}^n$ (graph edges denoted with $\sim$)
$\beta_r \in \mathbb{R}^+$, Laplacian-based regularization parameter
$\beta_s \in \mathbb{R}^+$, L_1 regularization parameter
$k' \leq K$, maximum allowable number of basis functions

Output: $\mathcal{I}$, set of selected basis functions (indices into ϕ)
$w_{\mathcal{I}}$, weight vector such that $\hat{V}(s) = \phi_{\mathcal{I}}(s)^T w_{\mathcal{I}}$

$c \leftarrow \sum_{i=1}^n \rho(s_i)(\phi(s_i) - \xi\gamma\phi(s'_i))r_i$
$\left[\bar{\beta}_s,\, j^*\right] \leftarrow \left[\max, \operatorname{argmax}\right]_j (|c_j|)$
Initialize active set $\mathcal{I} \leftarrow \{j^*\}$, $w \leftarrow \mathbf{0}$

while $(\bar{\beta}_s > \beta_s)$ and $(|\mathcal{I}| \leq k')$ and (Bellman residual not converged) **do**

1. Compute weight update direction $\Delta w_{\mathcal{I}}$:
$\Delta w_{\mathcal{I}} \leftarrow \hat{A}_{\mathcal{I},\mathcal{I}}^{-1} \operatorname{sign}(c_{\mathcal{I}})$
where: $\hat{A}_{\mathcal{I},\mathcal{I}} \leftarrow \sum_{i=1}^n \rho(s_i)[(\phi_{\mathcal{I}}(s_i) - \xi\gamma\phi_{\mathcal{I}}(s'_i))(\phi_{\mathcal{I}}(s_i) - \gamma\phi_{\mathcal{I}}(s'_i))^T + \ldots$
$\beta_r g_{\mathcal{I}}(s_i) g_{\mathcal{I}}(s_i)^T]$
$g(s_i) \leftarrow L(s_i, s_i)\,\phi(s_i)$
$g(s_i) \leftarrow g(s_i) + L(s_i, s_{nbr})\,\phi(s_{nbr}) \qquad \forall\{s_{nbr} | s_{nbr} \neq s \wedge s \sim s_{nbr}\}$
2. Compute correlation update direction Δc:
$\Delta c \leftarrow \sum_{i=1}^n \rho(s_i)[(\phi(s_i) - \xi\gamma\phi(s'_i))\,(\phi_{\mathcal{I}}(s_i) - \gamma\phi_{\mathcal{I}}(s'_i))^T \Delta w_{\mathcal{I}} + \ldots$
$\beta_r\, g(s_i) g_{\mathcal{I}}\,(s_i)^T \Delta w_{\mathcal{I}}]$
3. Find step size to add element to active set:
$[\alpha^*,\, j^*] \leftarrow \left[\min{}^+, \operatorname{argmin}\right]_{j \notin \mathcal{I}} \left(\frac{c_j - \bar{\beta}_s}{\Delta c_j - 1},\, \frac{c_j + \bar{\beta}_s}{\Delta c_j + 1}\right)$
4. Find step size to remove element from active set:
If (using LARS-FP), $\alpha^\# \leftarrow \infty$
Else, $\left[\alpha^\#,\, j^\#\right] \leftarrow \left[\min{}^+, \operatorname{argmin}\right]_{j \in \mathcal{I}} \left(-\frac{w_j}{\Delta w_j}\right)$
5. Update $\bar{\beta}_s, w_{\mathcal{I}}, c$:
$\alpha \leftarrow \min(\alpha^*, \alpha^\#, \bar{\beta}_s - \beta_s)$
$\bar{\beta}_s \leftarrow \bar{\beta}_s - \alpha, \quad w_{\mathcal{I}} \leftarrow w_{\mathcal{I}} + \alpha \Delta w_{\mathcal{I}}, \quad c \leftarrow c - \alpha \Delta c$
6. Adjust active set:
If $(\alpha^* < \alpha^\#)$, $\mathcal{I} \leftarrow \mathcal{I} \cup \{j^*\}$
Else, $\mathcal{I} \leftarrow \mathcal{I} - \{j^\#\}$

end while
OPTIONAL: $w_{\mathcal{I}} \leftarrow \hat{A}_{\mathcal{I},\mathcal{I}}^{-1} \hat{b}_{\mathcal{I}}$ where: $\hat{b}_{\mathcal{I}} \leftarrow \sum_{i=1}^n \rho(s_i)(\phi_{\mathcal{I}}(s_i) - \xi\gamma\phi_{\mathcal{I}}(s'_i))r_i$

BIBLIOGRAPHY

[1] Agaev, R., and Chebotarev, P. On the spectra of nonsymmetric Laplacian matrices. *Linear Algebra and Its Applications 399* (2005), 157–168.

[2] Antos, A., Szepesvári, C., and Munos, R. Learning near-optimal policies with Bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning 71*, 1 (2008), 89–129.

[3] Baird, L. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Machine Learning* (1995), pp. 30–37.

[4] Bekas, Constantine, and Saad, Yousef. Computation of smallest eigenvalues using spectral Schur complements. *SIAM Journal on Scientific Computing 27* (2005), 458–481.

[5] Belkin, M, and Niyogi, P. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation 6*, 15 (2003), 1373–1396.

[6] Belkin, M., and Niyogi, P. Towards a theoretical foundation for Laplacian-based manifold methods. *Journal of Computer and System Sciences 74*, 8 (2008), 1289–1308.

[7] Belkin, M., Niyogi, P., Sindhwani, V., and Bartlett, P. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research 7* (2006), 2399–2434.

[8] Bellman, R. *Introduction to Matrix Analysis*, 2nd ed. McGraw-Hill Education, New York, NY, 1970.

[9] Bennighof, Jeffrey K., and Lehoucq, R.B. An automated multilevel substructuring method for eigenspace computation in linear elastodynamics. *SIAM Journal on Scientific Computing 25* (2004), 2084–2106.

[10] Bertsekas, D. Differential training of rollout policies. In *Proceedings of the 35th Allerton Conference on Communication, Control, and Computing* (1997).

[11] Bertsekas, D., and Castañon, D. Adaptive aggregation methods for infinite horizon dynamic programming. *IEEE Transactions on Automatic Control 34* (1989), 589–598.

[12] Bertsekas, D., and Tsitsiklis, J. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

[13] Billera, L., and Diaconis, P. A geometric interpretation of the Metropolis-Hasting algorithm. *Statist. Science 16* (2001), 335–339.

[14] Bjorck, A., and Golub, G. Numerical methods for computing angles between linear subspaces. *Mathematics of Computation 27*, 123 (1973), 579–594.

[15] Bowling, M., Ghodsi, A., and Wilkinson, D. Action respecting embedding. In *Proceedings of the 22nd International Conference on Machine Learning* (New York, NY, 2005), ACM Press, pp. 65–72.

[16] Boyan, J. Least-squares temporal difference learning. In *Proceedings of the 16th International Conference on Machine Learning* (San Francisco, CA, 1999), Morgan Kaufmann, pp. 49–56.

[17] Bradtke, S., and Barto, A. Linear least-squares algorithms for temporal difference learning. *Machine Learning 22*, 1-3 (1996), 33–57.

[18] Bremer, J., Coifman, R., Maggioni, M., and Szlam, A. Diffusion wavelet packets. *Applied and Computational Harmonic Analysis 21*, 1 (2006), 95–112.

[19] Chen, S., Donoho, D., and Saunders, M. Atomic decomposition by basis pursuit. *SIAM Journal of Scientific Computing 20*, 1 (1998), 33–61.

[20] Chung, F. *Spectral Graph Theory*. Number 92 in CBMS Regional Conference Series in Mathematics. American Mathematical Society, Providence, RI, 1997.

[21] Chung, F. Laplacians and the Cheeger inequailty for directed graphs. *Annals of Combinatorics 9* (2005), 1–19.

[22] Coifman, R., and Maggioni, M. Diffusion wavelets. *Applied and Computational Harmonic Analysis, Special Issue: Diffusion Maps and Wavelets 21*, 1 (2006), 53–94.

[23] Crites, B., and Barto, A. Improving elevator performance using reinforcement learning. In *Advances in Neural Information Processing Systems 8*, D. Touretzky, M. Mozer, and M. Hasselmo, Eds. MIT Press, Cambridge, MA, 1996, pp. 1017–1023.

[24] Csató, L., and Opper, M. Sparse on-line Gaussian processes. *Neural Computation 14* (2002), 641–668.

[25] Daubechies, I. *Ten Lectures on Wavelets (CBMS - NSF Regional Conference Series in Applied Mathematics)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1992.

[26] Dayan, P. The convergence of TD(λ) for general λ. *Machine Learning 8* (1992), 341–362.

[27] Dayan, P. Improving generalization for temporal difference learning - the successor representation. *Neural Computation 5*, 4 (1993), 613–624.

[28] de Lathauwer, L., de Moor, B., and Vandewalle, J. A multilinear singular value decomposition. *SIAM Journal on Matrix Analysis and Applications 21*, 4 (2000), 1253–1278.

[29] de Lathauwer, L., de Moor, B., and Vandewalle, J. On the best rank-1 and rank-$(R_1, R_2, \ldots, R_N)$ approximation of higher-order tensors. *SIAM Journal on Matrix Analysis and Applications 21*, 4 (2000), 1324–1342.

[30] Drineas, P., and Mahoney, M. On the Nyström method for approximating a Gram matrix for improved kernel-based learning. *Journal of Machine Learning Research 6* (2005), 2153–2175.

[31] Drummond, C. Accelerating reinforcement learning by composing solutions of automatically identified subtasks. *Journal of Artificial Intelligence Research 16* (2002), 59–104.

[32] Eckart, C., and Young, G. The approximation of one matrix by another of lower rank. *Psychometrika 1*, 3 (1936), 211–218.

[33] Efron, B., Hastie, T., Johnstone, I., and Tibshirani, R. Least angle regression. *Annals of Statistics 32* (2004), 407–451.

[34] Elssel, K., and Voss, H. An a priori bound for automated multilevel substructuring. *SIAM Journal on Matrix Analysis and Applications 28*, 2 (2006), 386–397.

[35] Engel, Y., Mannor, S., and Meir, R. The kernel recursive least squares algorithm. *IEEE Transactions on Signal Processing 52* (2003), 2275–2285.

[36] Ernst, D., Geurts, P., and Wehenkel, L. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research 6* (2005), 503–556.

[37] Foster, D., and Dayan, P. Structure in the space of value functions. *Machine Learning 49* (2002), 325–346.

[38] Gao, W., Li, X., Yang, C., and Bai, Z. An implementation and evaluation of the AMLS method for sparse eigenvalue problems. *ACM Transactions on Mathematical Software 34*, 4 (2008), 1–27.

[39] Golub, G., and Van Loan, C. *Matrix Computations*, 3rd ed. Johns Hopkins University Press, Baltimore, MD, 1996.

[40] Hadamard, J. Sur les problèmes aux dérivées partielles et leur signification physique. *Princeton University Bulletin* (1902), 49–52.

[41] Hein, M., Audibert, J., and von Luxburg, U. From graphs to manifolds - weak and strong pointwise consistency of graph Laplacians. In *Proceedings of the 18th Conference on Learning Theory* (2005), pp. 470–485.

[42] Hinton, G., Osindero, S., and Teh, Y. A fast learning algorithm for deep belief nets. *Neural Computation 18* (2006), 1527–1554.

[43] Hurty, W. Dynamic analysis of structural systems using component modes. *American Institute of Aeronautics and Astronautics Journal 3*, 4 (1965), 678–685.

[44] Johns, J., and Mahadevan, S. Constructing basis functions from directed graphs for value function approximation. In *Proceedings of the 24th International Conference on Machine Learning* (New York, NY, 2007), ACM Press, pp. 385–392.

[45] Johns, J., and Mahadevan, S. Sparse approximate policy evaluation using graph-based basis functions. Tech. Rep. UM-CS-2009-041, University of Massachusetts Amherst, 2009.

[46] Johns, J., Mahadevan, S., and Wang, C. Compact spectral bases for value function approximation using Kronecker factorization. In *Proceedings of the 22nd National Conference on Artificial Intelligence* (2007), pp. 559–564.

[47] Johns, J., Osentoski, S., and Mahadevan, S. Representation discovery in planning using harmonic analysis. In *Proceedings of the AAAI Fall Symposium on Computational Approaches to Representation Change During Learning and Development* (2007), pp. 24–31.

[48] Johns, J., Petrik, M., and Mahadevan, S. Hybrid least-squares algorithms for approximate policy evaluation. *Machine Learning 76*, 2 (2009), 243–256.

[49] Karypis, G., and Kumar, V. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing 20*, 1 (1999), 359–392.

[50] Keller, P., Mannor, S., and Precup, D. Automatic basis function construction for approximate dynamic programming and reinforcement learning. In *Proceedings of the 23rd International Conf. on Machine Learning* (2006), pp. 449–456.

[51] Koller, D., and Parr, R. Policy iteration for factored MDPs. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence* (2000), Morgan Kaufmann, pp. 326–334.

[52] Kolter, J., and Ng, A. Regularization and feature selection in least-squares temporal difference learning. In *Proceedings of the 26th International Conference on Machine Learning* (2009), pp. 521–528.

[53] Konidaris, G., and Osentoski, S. Value function approximation in reinforcement learning using the Fourier basis. Tech. Rep. TR-2008-19, University of Massachusetts, Department of Computer Science, 2008.

[54] Kostrykin, V., Makarov, K. A., and Motovilov, A. K. On a subspace perturbation problem. In *Proc. of the American Mathematical Society* (2003), vol. 131, pp. 1038–1044.

[55] Kretchmar, R., and Anderson, C. Using temporal neighborhoods to adapt function approximators in reinforcement learning. In *Interational Work Conference on Artificial and Natural Neural Networks* (1999), pp. 488–496.

[56] Lagoudakis, M., and Parr, R. Least-squares policy iteration. *Journal of Machine Learning Research 4* (2003), 1107–1149.

[57] Lagoudakis, M., Parr, R., and Littman, M. Least-squares methods in reinforcement learning for control. In *Proceedings of the Second Hellenic Conference on Artificial Intelligence* (2002), pp. 249–260.

[58] Langville, Amy N., and Stewart, William J. A Kronecker product approximate preconditioner for SANs. *Numerical Linear Algebra with Applications 11,* 8 (2004), 723–752.

[59] Lee, H., Battle, A., Raina, R., and Ng, A. Efficient sparse coding algorithms. In *Advances in Neural Information Processing Systems* (2007), vol. 19, pp. 801–808.

[60] Lewicki, M., and Sejnowski, T. Learning overcomplete representations. *Neural Computation 12,* 2 (2000), 337–365.

[61] Li, L. A worst-case comparison between temporal difference and residual gradient with linear function approximation. In *Proceedings of the 25th International Conference on Machine Learning* (2008), pp. 560–567.

[62] Loth, M., Davy, M., and Preux, P. Sparse temporal difference learning using LASSO. In *IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning* (2007), pp. 352–359.

[63] Maggioni, M., and Mahadevan, S. A multiscale framework for Markov decision processes using diffusion wavelets. Tech. Rep. TR-2006-36, University of Massachusetts, Department of Computer Science, 2006.

[64] Mahadevan, S. Representation policy iteration. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence* (2005), pp. 372–379.

[65] Mahadevan, S. Learning representation and control in Markov decision processes: New frontiers. *Foundations and Trends in Machine Learning 1,* 4 (2009), 403–565.

[66] Mahadevan, S., and Maggioni, M. Proto-value functions: A Laplacian framework for learning representation and control in Markov decision processes. Tech. Rep. TR-2006-35, University of Massachusetts, 2006.

[67] Mahadevan, S., and Maggioni, M. Proto-value functions: A Laplacian framework for learning representation and control in Markov decision processes. *Journal of Machine Learning Research 8* (2007), 2169–2231.

[68] Mahadevan, S., Maggioni, M., Ferguson, K., and Osentoski, S. Learning representation and control in continuous Markov decision processes. In *Proceedings of the 21st National Conference on Artificial Intelligence* (2006).

[69] Mallat, S., and Zhang, Z. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing 41*, 12 (1993), 3397–3415.

[70] Menache, I., Mannor, S., and Shimkin, N. Basis function adaptation in temporal difference reinforcement learning. *Annals of Operation Research 134*, 1 (2005), 215–238.

[71] Merkwirth, C., Parlitz, U., and Lauterborn, W. Fast nearest neighbor searching for nonlinear signal processing. *Physical Review E 62*, 2 (2000), 2089–2097.

[72] Merris, R. *Multilinear Algebra*. CRC Press, 1997.

[73] Moghaddam, B., Gruber, A., Weiss, Y., and Avidan, S. Sparse regression as a sparse eigenvalue problem. In *Information Theory and Applications Workshop* (2008), pp. 121–127.

[74] Munos, R. Error bounds for approximate policy iteration. In *Proceedings of the 20th International Conference on Machine Learning* (2003), pp. 560–567.

[75] Natarajan, B. Sparse approximate solutions to linear systems. *SIAM Journal on Computing 24*, 2 (1995), 227–234.

[76] Ng, A., Jordan, M., and Weiss, Y. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems 14* (2002), pp. 849–856.

[77] Olshausen, B., and Field, D. Emergence of simple-cell receptive fields by learning a sparse code for natural images. *Nature 381*, 6583 (1996), 607–609.

[78] Osentoski, S. *Action-Based Representation Discovery in Markov Decision Processes*. PhD thesis, University of Massachusetts Amherst, 2009.

[79] Osentoski, S., and Mahadevan, S. Learning state-action basis functions for hierarchical MDPs. In *Proceedings of the 24th International Conference on Machine Learning* (New York, NY, 2007), ACM Press, pp. 705–712.

[80] Page, L., Brin, S., Motwani, R., and Winograd, T. The PageRank citation ranking: Bringing order to the web. Tech. rep., Stanford Digital Library Technologies Project, 1998.

[81] Parr, R., Painter-Wakefield, C., Li, L., and Littman, M. Analyzing feature generation for value-function approximation. In *Proceedings of the 24th International Conference on Machine Learning* (New York, NY, 2007), ACM Press.

[82] Pati, Y., Rezaiifar, R., and Krishnaprasad, P. Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Proceedings of the 27th Annual Asilomar Conference on Signals, Systems, and Computers* (1993), pp. 40–44.

[83] Petrik, M. An analysis of Laplacian methods for value function approximation in MDPs. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence* (2007), pp. 2574–2579.

[84] Pitsianis, N. *The Kronecker Product in Approximation and Fast Transform Generation*. PhD thesis, Department of Computer Science, Cornell University, Ithaca, NY, 1997.

[85] Puterman, M. L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, 1st ed. John Wiley and Sons, New York, NY, 1994.

[86] Roweis, S., and Saul, L. Nonlinear dimensionality reduction by local linear embedding. *Science 290* (2000), 2323–2326.

[87] Samuel, A. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development 3* (1959), 210–229.

[88] Schoknecht, R. Optimality of reinforcement learning algorithms with linear function approximation. In *Advances in Neural Information Processing Systems 15* (2003), pp. 1555–1562.

[89] Schweitzer, P., and Seidmann, A. Generalized polynomial approximations in Markovian decision processes. *Journal of Mathematical Analysis and Applications 110* (1985), 568–582.

[90] Simon, H. *The Sciences of the Artificial*, 3rd ed. MIT Press, Cambridge, MA, USA, 1996.

[91] Smart, W. Explicit manifold representations for value-function approximation in reinforcement learning. In *Proceedings of the 8th International Symposium on AI and Mathematics* (2004).

[92] Smola, A., and Kondor, R. Kernels and regularization on graphs. In *Proceedings of the 16th Annual Conference on Learning Theory* (2003).

[93] Smola, A., and Schölkopf, B. Sparse greedy matrix approximation for machine learning. In *Proceedings of the 17th International Conference on Machine Learning* (2000), Morgan Kaufmann, pp. 911–918.

[94] Sugiyama, M., Hachiya, H., Towell, C., and Vijayakumar, S. Value function approximation on non-linear manifolds for robot motor control. In *Proceedings of the IEEE International Conference on Robotics and Automation* (2007), pp. 1733–1740.

[95] Sugiyama, M., Hachiya, H., Towell, C., and Vijayakumar, S. Geodesic Gaussian kernels for value function approximation. *Autonomous Robots 25* (2008), 287–304.

[96] Sutton, R. Learning to predict by the methods of temporal differences. *Machine Learning 3* (1988), 9–44.

[97] Sutton, R. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8* (Cambridge, MA, 1996), D. Touretzky, M. Mozer, and M. Hasselmo, Eds., MIT Press, pp. 1038–1044.

[98] Sutton, R., and Barto, A. *Reinforcement Learning*. MIT Press, Cambridge, MA, 1998.

[99] Sutton, R., Maei, H., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, C., and Wiewiora, E. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the 26th International Conference on Machine Learning* (2009), pp. 993–1000.

[100] Szita, I., and Lörincz, A. Learning Tetris using the noisy cross-entropy method. *Neural Computation 18*, 12 (2006), 2936–2941.

[101] Tenenbaum, J. B., de Silva, V., and Langford, J. C. A global geometric framework for nonlinear dimensionality reduction. *Science 290*, 5500 (2000), 2319–2323.

[102] Tesauro, G. Temporal difference learning and TD-Gammon. *Communications of the ACM 38*, 3 (1995), 58–68.

[103] Tibshirani, R. Regression shrinkage and selection via the LASSO. *Journal of the Royal Statistical Society, Series B 58* (1996), 267–288.

[104] Tikhonov, A., and Arsenin, V. *Solutions of Ill-Posed Problems*. Wiley, New York, 1977.

[105] Utgoff, P., and Precup, D. Relative value function approximation. Tech. Rep. UM-CS-1997-003, University of Massachusetts Amherst, 1997.

[106] Watkins, C. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, 1989.

[107] Weaver, L., and Baxter, J. Reinforcement learning from state and temporal differences. Tech. rep., Australian National University, 1999.

[108] Williams, C., and Seeger, M. Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems 13* (2001), T. Leen, T. Diettrich, and V. Tresp, Eds., pp. 682–688.

[109] Williams, R., and Baird, L. Tight performance bounds on greedy policies based on imperfect value functions. Tech. Rep. NU-CCS-93-14, Northeastern University, November 1993.

[110] Yang, C. Solving large-scale eigenvalue problems in SciDAC applications. *Journal of Physics: Conference Series 16* (2005), 425–434.